Approches visuelles pour l'amélioration de la présence en réalité virtuelle

François De Sorbier de Pougnadoresse

Université Paris-Est

THÈSE
pour obtenir le grade de
Docteur de l'Université Paris-Est

Spécialité : Informatique

Présentée et soutenue publiquement par
François de Sorbier de Pougnadoresse
le 27 novembre 2008

APPROCHES VISUELLES POUR L'AMÉLIORATION DE LA PRÉSENCE EN RÉALITÉ VIRTUELLE

VISUAL APPROACHES TO INCREASE PRESENCE IN VIRTUAL REALITY

Directeur de Thèse
Gilles Bertrand

Jury

Rapporteurs :	Jean-Pierre Jessel	Professeur à l'université de Toulouse
	Yannick Remion	Professeur à l'université de Reims
Examinateurs :	Gilles Bertrand	Professeur à Université Paris-Est, ESIEE, IGM
	Guillaume Moreau	Maître de conférence à l'École Centrale Nantes
	Anatole Lécuyer	Chercheur à l'INRIA Rennes
	Venceslas Biri	Maître de conférence à l'Université Paris-Est

à mes parents,
à mes sœurs,

Remerciements

Je voudrais tout d'abord adresser mes remerciements à Jean-Pierre Jessel, Yannick Remion, Guillaume Moreau et Anatole Lécuyer pour avoir accepté de faire partie de mon jury de thèse.

Je remercie tout particulièrement Gilles Bertrand qui m'a accueilli dans son équipe et a accepté de devenir mon directeur de thèse au cours de mon doctorat. J'ai pu, de cette manière, continuer sereinement mes travaux de recherche.

Je voudrais également adresser ma gratitude à Didier Arquès et Benoît Piranda qui m'ont offert la possibilité de commencer une thèse au sein de l'équipe mais également pour m'avoir fait apprécier l'enseignement avec l'école d'ingénieur IMAC.

Mes plus vifs remerciements vont à Venceslas Biri qui, pendant toutes ces années, m'a encouragé, soutenu et a toujours été de très bon conseil. Merci pour toute cette confiance que tu as eue en moi.

J'adresse aussi un très grand merci à toutes les personnes que j'ai pu côtoyer durant ces quatre années et qui m'ont toujours offert leur sympathie, leur bonne humeur et leur soutien. Mes remerciements vont plus particulièrement à Vincent Nozick, Patrice Bouvier, Pascal Chaudeyrac, Olivier Derpierre, Cyril Pichard, Ismaël Daribo, Benjamin Raynal et les étudiants IMAC de la promo 2004-2007 avec lesquels j'ai passé des moments inoubliables.

Enfin, je voudrais remercier ma famille pour tout le soutient qu'elle m'a apporté et tout ce qu'elle a pu faire pour moi.

Résumé

Mots-clés: réalité virtuelle, présence, immersion visuelle, stéréoscopie, flou, avatar

Le sentiment de présence, but ultime de la réalité virtuelle, peut être atteint en stimulant ces quatre «piliers» que sont l'immersion, l'interaction, le maintien de la boucle action-perception et les émotions. Notre objectif est de proposer des méthodes visant à améliorer ce sentiment en s'intéressant plus particulièrement à la perception visuelle.

Dans cette optique, nous proposons tout d'abord une solution appliquant le rendu stéréoscopique sur carte graphique. Traditionnellement effectué en deux passes, ce rendu se fait maintenant en une passe, grâce aux *shaders* et au regroupement de certaines phases de calculs. Nous étendons ce processus de rendu aux tout récents écrans auto-stéréoscopiques nécessitant plus de deux vues, améliorant d'autant plus les temps de calcul. Pour assurer l'immersion et l'interaction, voire l'émotion, nous avons aussi cherché à diminuer la fatigue oculaire induite par les images stéréoscopiques, en ajoutant un flou de profondeur de champ. Ce flou, obtenu en temps réel grâce aux *shaders*, permet également d'inviter l'observateur à focaliser son attention sur des objets précis au lieu de laisser son regard errer.

Enfin, un objectif pour obtenir le sentiment de présence est de faire croire à l'utilisateur qu'il existe dans la scène virtuelle. Notre contribution à ce but, est d'intégrer de manière naturelle une représentation virtuelle de l'utilisateur. Pour cela, nous créons par *visual hulls* un avatar à l'aide de caméras. Finalement, cet avatar est employé pour illustrer la présence de l'utilisateur au travers de surfaces réfléchissantes virtuelles ou de la projection de son ombre.

Abstract

Keywords: virtual reality, presence, visual immersion, stereoscopy, blur, avatar

The feeling of presence is the ultimate goal of a virtual reality system. It can be achieved by stimulating its four pillars that are immersion, interaction, consistency of the action-perception loop and emotions. Our objective is submit methods which could increase this feeling of presence. We will especially focus on visual perception.

First, we present a solution applying the stereoscopic rendering on GPU. Using the latest shaders extensions, we join the traditional two passes to a single that reduce redundant computation. Our method can also be applied on recent autosterescopic screens which need about ten different view points. The second contribution refers to eyestrain caused when a user watches stereoscopic images during a long time. We propose to use the graphic cards to render, in real time, a blur on problematic areas thus the user will focus on the others. We also use blur to point out some objects of the virtual scene that we considerate as important and avoid to let user's eye wander.

Finally, we propose a method to increase the presence of the user into the virtual environment. An avatar, based on images taken from several cameras and visual hull method, is first build. In a second stage, the avatar is integrated into the virtual scene so it seems natural to the user. Thus we use reflective surfaces like mirrors or virtual lights interactions like shadows.

Table des matières

Table des figures

Introduction

Depuis plusieurs années le domaine de la réalité virtuelle ne cesse de connaitre un essor de plus en plus important. La plupart des laboratoires s'équipent de telles installations tandis que les industriels continuent d'améliorer l'efficacité des équipements qu'ils ont pu acquérir. De plus, les systèmes de réalité virtuelle tendent à s'ouvrir au tout à chacun, mais sous une forme certes plus simplifiée que celle que nous pouvons croiser dans la recherche.

La réalité virtuelle a l'avantage de pouvoir bénéficier des nombreux perfectionnements issus des différentes spécialités qui la composent comme la synthèse d'images, le son, les sciences cognitives ou l'électronique... Au final les installations produisent des résultats de plus en plus crédibles au point de tromper nos perceptions et participent ainsi à créer le sentiment de présence chez son utilisateur. Ce but ultime de la réalité virtuelle vise à faire croire à l'utilisateur qu'il appartient à l'univers virtuel qui lui est présenté tout en ayant le sentiment que cet espace virtuel existe vraiment.

Les recherches concernant la création de ce sentiment de présence sont multiples et ont permis le développement de nombreux algorithmes et interfaces aidant à l'immersion de l'utilisateur. Plus les sens de ce dernier sont stimulés, plus l'immersion sera efficace. Dans notre cas nous avons choisi de nous concentrer sur la recherche de solutions qui pourraient améliorer la présence à l'aide d'indices visuels. Pour le moment la plupart des propositions s'orientent vers des contenus virtuels de plus en plus réalistes, soit en donnant plus de consistance aux images en utilisant par exemple des méthodes de stéréoscopie soit en améliorant leur photo-réalisme. Cependant, ces solutions proposent rarement d'améliorer la perception que nous avons des contenus virtuels ou alors le sentiment que nous appartenons d'un point de vue visuel à cet environnement.

Dans ce mémoire, nous proposons dans un premier temps deux outils visant à améliorer la perception des images produites dans les installations de réalité virtuelle. Notre première approche consiste à accroître la rapidité de génération des images stéréoscopiques qui, jusqu'à présent, se fondait principalement sur la nécessité de produire un double rendu de la scène depuis deux points de vue légèrement décalés. À l'aide des dernières optimisations des cartes graphiques, il est maintenant possible de manipuler les différentes primitives ce qui nous permet d'unifier les traitements effectués sur les sommets et ainsi de diminuer les temps des rendus. Enfin l'apparition des procédés auto-stéréoscopiques, qui permettent de voir des images en relief sans avoir à porter d'interfaces spécifiques, né-

cessitent plus de deux images et l'application de notre méthode permet alors de conserver un affichage en temps réel.

Notre seconde contribution a pour but d'aider à limiter certains des effets induits par les images stéréoscopiques. Il est courant de constater l'apparition de fatigue chez un utilisateur, engendrée par une difficulté à fusionner certaines parties de l'image. Pour limiter ces inconvénients nous proposons d'inciter ce dernier à focaliser son regard vers des zones spécifiques de l'image en appliquant un flou de profondeur de champ sur celles qu'il ne doit pas regarder. L'emploi de cette méthode nous permet également de diriger l'attention de l'utilisateur vers certains objets de la scène qui sont considérés comme d'intérêt, au lieu de laisser vagabonder le regard de l'utilisateur dans l'image.

Dans la troisième partie nous décrivons une méthode qui permet d'aider à développer le sentiment de présence chez l'utilisateur, en lui donnant l'impression qu'il «existe» dans l'environnement virtuel auquel il participe. À cette fin, nous mettons en œuvre des moyens d'identification naturels comme l'utilisation de surfaces réfléchissantes ou la projection de l'ombre de l'utilisateur. L'une des méthodes possibles pour obtenir ce résultat consiste à effectuer une reconstruction en temps réel de l'utilisateur à l'aide de caméras. Plus particulièrement, nous employons la méthode des *visual hulls* qui se fonde sur la détection de la silhouette de l'utilisateur depuis différents points de vue repartis autour de lui, puis du calcul de l'intersection des cônes générés en conséquence et enfin le rendu. De cette façon nous obtenons une copie virtuelle en temps réel de l'utilisateur que nous pouvons exploiter pour accroître le sentiment de présence en nous aidant uniquement de la partie visuelle du système de réalité virtuelle.

Les travaux présentés dans ce mémoire ont fait l'objet de publications. La partie sur le rendu stéréoscopique par carte graphique a été publié dans de Sorbier, Nozick et Biri [dSNB08a, dSNB08b]. Quant à la méthode d'utilisation du flou dans les images stéréoscopiques, elle a été publiée dans de Sorbier *et al.* [PdS04, PdSA05]. En ce qui concerne la dernière partie, nous désirons avant toute publication achever la définition de notre protocole d'évaluation du sentiment de présence ainsi que les différentes phases de mise en application afin de vérifier nos hypothèses.

Première partie

Réalité virtuelle

1

Définition de la réalité virtuelle

La réalité virtuelle est un domaine assez complexe à cerner et dont l'objectif, consistant à faire en sorte que l'utilisateur est l'impression d'appartenir à un environnement virtuel, est difficile à atteindre. Dans cette partie nous allons essayer de décrire les différentes approches qui ont apporté plusieurs définitions de la réalité virtuelle. Nous en profiterons également pour proposer notre point de vue sur un des fondements de la réalité virtuelle : le sentiment de présence.

1.1 Origines du terme

Réalité virtuelle ou RV, voilà un mot qui au premier abord peut paraître bien étrange à cause de l'opposition des deux termes en contradiction. Cet oxymoron se compose donc d'un côté du nom *réalité* et de l'autre de l'adjectif *virtuelle* qui tend à supposer qu'il pourrait exister un univers réel généré via un ordinateur. En fait la réalité virtuelle a une origine anglaise, *virtual reality*, dont le premier emploi est associé à Jaron Lanier qui l'a utilisé dans les années 80. La première définition propagée par ce dernier, décrit ainsi la réalité virtuelle comme une réalité synthétisée partageable avec d'autres personnes, qui peut être appréhendée par nos sens et avec laquelle nous pouvons interagir, le tout par l'intermédiaire d'une interface [Hei88].

Un autre terme faisant référence à ce que désignait «virtual reality» fut toutefois donné en 1983 par Myron Krueger dans son livre «Artificial Reality» [Kru89].

C'est dans les années 90 que le terme réalité virtuelle fait son entrée en France et bénéficie alors d'une forte médiatisation qui contribue à imposer son utilisation. Tout le monde n'est cependant pas d'accord sur l'emploi de cette traduction littérale. Ainsi Papin fait très justement remarquer qu'une traduction possible du mot anglais *virtual* peut être «quasiment» ce qui l'amène à proposer le terme de réalité vicariante et qui se rapproche peut-être plus de ce que cherchait à définir Jaron Lanier.

Le terme réalité virtuelle ne doit toutefois pas être confondu avec *réalité augmentée*

qui consiste en l'intégration d'éléments synthétiques dans des images d'environnements réels. Un *environnement virtuel* est également un terme qui ne doit pas être confondu avec réalité virtuelle car il ne désigne qu'un contenu généré par ordinateur et qui est donc bien trop restrictif pour définir ce qu'est la RV.

1.2 Réalité virtuelle : définition

Au cours des dernières années les définitions données à la réalité virtuelle se sont succédé. Étant donné qu'il s'agissait de mettre un principe sur un «domaine» à la conjonction de plusieurs spécialités, chacun y est allé de sa proposition jusqu'à ce qu'une convergence se réalise.

Pour le tout à chacun, l'image de la réalité virtuelle a été fortement véhiculée par les médias qui ont tendance à faire l'apologie d'une technologie plutôt que de s'intéresser à un tout. Il est alors courant de rencontrer des personnes pour qui la réalité virtuelle consiste à regarder un film en relief ou à porter un casque tout droit sorti d'un film de science fiction. C'est d'ailleurs l'industrie du cinéma qui a participé à donner une image fantaisiste et voire même surréaliste de la réalité virtuelle avec les films comme *Le Cobaye*(1992), *eXistenZ*(1999) ou encore *Matrix*(1999). Malgré cela, on peut constater que certaines idées sur la réalité virtuelle, comme les interfaces imaginées pour le film *Minority Report*(2002), commencent à réellement émerger tel que le «*Sensitive Wall*» [iO].

Outre la définition donnée par Jaron Lanier [Hei88] qu'il qualifia lui-même plus tard de délirante, il est possible de trouver des définitions de la réalité virtuelle donnant un aperçu de ce que peut être ce domaine :

- Système qui permet à un ou plusieurs utilisateurs de regarder, de se déplacer et de réagir dans un univers simulé par ordinateur (Encyclopédie *Encarta*).
- Simulation d'un environnement réel par des images de synthèse (*Larousse*).
- Simulation informatique interactive immersive, visuelle, sonore et/ou haptique, d'environnements réels ou imaginaires (*Wikipédia*).
- Moyen pour les êtres humains de visualiser, manipuler et interagir avec des ordinateurs et des données complexes [AB92].
- Illusion de participer à un environnement synthétique plutôt que d'en être un observateur extérieur. La réalité virtuelle dépend de la 3D, de la stéréoscopie, du *tracking* et du son binaurale. La réalité virtuelle est une expérience immersive et multi-sensorielle [Tha93].
- Permet de naviguer et visualiser un monde en trois dimensions en temps réel avec six degrés de liberté : la liberté de se déplacer, de regarder en avant ou en arrière, à droite ou à gauche, ou en haut et en bas [SS95].

Cette liste de définitions donne un aperçu plus ou moins exact de la réalité virtuelle. Généralement la terminologie utilisée a tendance à être trop superficielle ou alors bien

trop ciblée sur un aspect donné de la RV. La définition la plus complète et qui décrit le mieux la réalité virtuelle est donnée dans «Le Traité de la réalité virtuelle» [AFT03] :

La réalité virtuelle est un domaine scientifique et technique exploitant l'informatique et des interfaces comportementales en vue de simuler dans un monde virtuel le comportement d'entités 3D, qui sont en interaction en temps réel entre elles et avec un ou des utilisateurs en immersion pseudo-naturelle par l'intermédiaire de canaux sensorimoteurs.

1.3 Finalité de la réalité virtuelle

Pour se donner une autre idée de ce qu'est la réalité virtuelle on peut décrire sa finalité. Le but ultime de la réalité virtuelle est de générer chez son utilisateur un sentiment de présence. Le sens donné à ce terme est plus ou moins variable dans les domaines de recherche qui s'y consacrent. Ainsi, parmi les différentes théories qui ont pu être exposées [She92, SvdSKvdM01, RDI03, WW03a], deux s'en détachent qui ont été plusieurs fois reprises et qui tendent à se généraliser. Dans la première, Lombard & Ditton [LD97] proposent de définir la présence comme «*perceptual illusion of non-mediation*», c'est-à-dire que le sentiment de présence apparaît lorsque l'utilisateur n'a plus conscience que l'environnement virtuel est présenté sur un support physique. La deuxième théorie de Slater et Steed [SS00], qui sera par la suite conservée comme définition de la présence, désigne cette dernière comme le «*sense of being there*» qui peut se traduire comme avoir l'impression d'appartenir complètement à l'univers virtuel qui est présenté et non plus à celui dans lequel on se trouve réellement.

Il y a quatre conditions nécessaires, sur lesquelles une expérience doit reposer, pour provoquer le sentiment de présence :

- l'immersion ;
- l'interaction ;
- le maintien de la boucle action-perception ;
- la création d'émotions.

L'immersion [SMDW98], qui peut être source de confusions avec le terme présence, a pour but de stimuler le maximum de sens de l'utilisateur d'un système de réalité virtuelle. En plus des sens les plus connus que sont la vue, l'ouïe, l'odorat, le goût et le toucher, il y a d'autres sens qu'il est possible de stimuler comme la kinesthésie ou la proprioception. La kinesthésie définie les sensations perçues au niveau musculaire lors d'un mouvement. La proprioception fait référence à la perception consciente ou non de la position relative des différentes parties de notre corps. Ces deux derniers sens ont leur importance, par exemple, dans la conception d'un simulateur de conduite qui nécessite de faire ressentir à l'utilisateur des forces type accélérations ou centripètes et qui doivent être reproduites alors qu'il n'y a pas de réel mouvement de la «cabine» [Dag05].

L'interaction [Min95] consiste à faire interagir l'utilisateur avec l'environnement virtuel qui lui est présenté. Les actions autorisées dans cet univers peuvent permettre à l'utilisateur de se déplacer, manipuler un objet virtuel, etc. Mais l'environnement provoque aussi une réaction chez l'utilisateur. Ces dernières ne se font pas seulement d'un point de vue physique via l'utilisation d'interfaces dédiées, mais elles peuvent se faire également d'un point de vue visuel, auditif et même émotionnel.

Le maintien de la boucle action-perception [DSG94, Fus04] ou temps réel (à ne pas confondre avec un affichage à environ 30 images par seconde) consiste à faire en sorte que les actions effectuées par l'utilisateur aient une réaction naturelle dans l'environnement virtuel. Un exemple est celui d'un simulateur de sport comme le tennis : le joueur ayant à sa disposition une raquette de tennis va frapper une balle virtuelle et celle-ci doit réagir de manière cohérente en fonction de la force induite par le coup ainsi que de sa direction. Si la balle n'a pas la réaction escomptée alors la boucle action-perception sera brisée ce qui aura pour conséquence de diminuer le sentiment de présence de l'utilisateur.

Enfin la génération d'émotions est importante dans la création d'un sentiment de présence comme mentionné dans [HA99] et [BBA+04]. Le but est de capter l'attention en utilisant un contexte virtuel propice à générer des émotions pouvant susciter un intérêt chez l'utilisateur. Autrement dit, on cherche à ce que ce dernier arrive à oublier qu'il est au centre d'un système de réalité virtuelle composé d'éléments perturbateurs tels que les ordinateurs, l'écran, les câbles qui pourraient venir rompre le sentiment de présence. Le processus est assez similaire à celui que l'on retrouve lors de la lecture d'un livre ou lors du visionnage d'un film. Concrètement, on peut prendre l'exemple d'un adepte du ski qui se retrouve captivé par un reportage sur les sports de glisse en montagne à cause du lien sentimental (souvenirs, expérience, peur,...) que la vidéo provoque et en arrive à oublier le contexte présent. De la même façon, une application de réalité virtuelle va chercher à focaliser l'attention d'un utilisateur en proposant un contenu riche en intérêt et qui peut passer par exemple par l'utilisation d'un scénario attractif.

Créer un système de réalité virtuelle, utilisant pleinement ces quatre conditions, n'est pas réalisable pour le moment en raison des nombreuses limites technologiques qui existent. Conduire une expérience de réalité virtuelle, en mettant l'accent sur une seule partie des conditions, est tout à fait possible et cela n'empêchera pas d'obtenir un sentiment de présence, mais dans une moindre mesure. Actuellement, la plupart des systèmes sont fondés sur une forte dominante définie autour de l'immersion, on trouve ensuite l'interaction, le maintien de la boucle action-perception et enfin la création d'émotions.

2

Domaines de recherche associés à la réalité virtuelle

Comme cela a déjà été évoqué dans le chapitre précédent, la réalité virtuelle n'est pas un domaine à part entière, mais se situe plutôt à l'intersection de multiples compétences provenant de différents domaines de recherche. La liste de ces derniers est assez vaste, car la création d'un système de réalité virtuel requiert de nombreuses compétences. L'énumération des différentes disciplines, que nous allons présenter, est issue de nos expériences sur la réalité virtuelle, acquissent ces dernières années.

De notre point de vue, nous pouvons distinguer six domaines principaux de recherche qui peuvent s'appliquer à la réalité virtuelle. Nous présenterons les domaines de l'image (composé de la synthèse d'images, du traitement d'image et de la vision par ordinateur), celui du son, de l'électronique, de l'intelligence artificielle, du réseau et enfin le domaine des sciences cognitives.

2.1 L'image

Le domaine de l'image peut également être considéré comme une réunion de plusieurs disciplines. En plus de l'affichage de données visuelles au travers, par exemple, de la synthèse d'images, il est parfois nécessaire de récupérer de l'information depuis ces images. Cela s'effectue par l'utilisation de méthodes de traitement d'images ou de vision par ordinateur qui peuvent, entre autres, reposer sur l'exploitation de flux vidéos.

La synthèse d'images ne se résume pas simplement à l'affichage d'un ensemble de triangles qui vise à constituer une scène réaliste ou non. Il faut effectivement que l'environnement virtuel puisse sembler suffisamment crédible à l'observateur soit d'un point de vue purement visuel soit sur le plan des performances. Au cours des dernières années, un grand nombre d'algorithmes ont été développés avec l'objectif d'obtenir un contenu visuellement réaliste avec par exemple une illumination plus proche du comportement de la lumière ou l'ajout de détails de plus en plus fins. L'évolution des matériels informatiques,

tels que les cartes graphiques ou les cartes physiques, ont certes permis de rendre accessible certaines de ces techniques en temps réel. Mais ce sont principalement les recherches sur les optimisations d'affichage avec des graphes de scène, la simplification de maillage, la modélisation mathématique et informatique de l'influence du milieu, *etc*, qui permettent de rendre une scène crédible en temps réel.

Outre les images de synthèse, plusieurs autres domaines liés à l'image, d'une manière générale, ont leur importance. Certaines applications de RV ont parfois besoin de connaître la position d'un utilisateur et appliquent par conséquent un *tracking* pouvant être fait à l'aide de caméras. Les différents flux vidéos récupérés nécessitent alors une décompression, un traitement voire une compression, afin d'extraire une donnée particulière telle qu'un marqueur ou une silhouette. On a alors besoin de faire appel à des connaissances liées au traitement d'images (voire traitement du signal) ou à la vision par ordinateur.

2.2 Le son

Le son en réalité virtuelle a une place très importante car c'est, à l'instar de l'image, un stimulus dominant de notre perception d'un environnement. Ce domaine nécessite encore beaucoup de recherches car, outre l'aspect simulation sur la création et la propagation d'un son dans un espace virtuel, il y a aussi la restitution dans le système de réalité virtuelle qui peut être complexe. La plupart des systèmes utilisent un ensemble d'enceintes car elles permettent une meilleure spatialisation du son, mais des études acoustiques sont alors nécessaires pour tout d'abord s'adapter à l'environnement dans lequel l'installation est placée (importance de la composition des matériaux, de l'isolation, ...) mais aussi pour faire en sorte que le système puisse être multi-utilisateurs. Dans ce cas l'objectif est de trouver la disposition la plus adaptée pour que la restitution sonore se fasse dans les meilleures conditions.

En plus de l'aspect purement matériel, c'est aussi la connaissance de l'impact physiologique d'un son qui est intéressant. Comme il est pour le moment assez difficile de pouvoir restituer un environnement sonore dans son intégralité, il faut pouvoir sélectionner les sons qui vont le refléter au mieux. De plus, il est tout à fait possible de déclencher des émotions en utilisant des sons biens spécifiques à des moments biens précis (par exemple le son émit par un moustique va susciter immédiatement de la crainte) ou alors sa capacité suggestive (diffuser quelques sons d'eau pour donner l'envie de boire).

2.3 L'électronique

Malgré les apparences, l'électronique a une place importante en réalité virtuelle. Ce domaine n'apparaît pas directement dans les installations, mais est forcement utilisé en raison de la présence des différentes interfaces. Des compétences en électronique permettent ainsi de créer de nouvelles interfaces (haptiques, de *tracking*, ...) expérimentales pour

pouvoir améliorer le sentiment de présence lors d'une expérience de réalité virtuelle. Il est alors possible d'en améliorer la précision, l'ergonomie, le bruit, *etc.* Un aperçu des différentes interfaces existantes en réalité virtuelle est présenté dans l'annexeřefannexe :inter.

2.4 L'intelligence artificielle

L'interaction et l'immersion dans un environnement virtuel peuvent nécessiter que certaines de ses composantes, comme des acteurs non-humains, aient leur propre manière de réagir en fonction des actions de l'utilisateur. Cette autonomie des agents virtuels est obtenue en employant des algorithmes appartenant au domaine de l'intelligence artificielle qui sont par exemple couramment utilisés pour le jeu vidéo.

2.5 Le réseau

Le réseau intervient en RV dans la création d'applications liées à un travail distant comme la téléopération (actions sur une machine distante) ou la téléprésence (réunion de personnes localisées dans des lieux géographiquement différents). En plus de la couche physique qui permet de transmettre des données d'un ordinateur à un autre (carte réseau, câblage, *switch etc...*), il y a une partie logiciel qui consiste notamment au choix des protocoles à utiliser, des informations à transmettre. Par exemple des descripteurs de contenus, comme le VRML (*Virtual Reality Markup Language*) ou l'«*eXtensible 3D*» [con] qui lui a succédé, ont été créés pour permettre la visualisation de mondes virtuels via Internet.

Certaines configurations de salles de réalité virtuelle peuvent nécessiter de gros moyens de calcul pour, par exemple, gérer plusieurs unités d'affichage ou contrôler divers périphériques comme des caméras. Dans ce cas, un PCs est censé centraliser les diverses informations et les transmettre vers les autres. La nécessité d'une connectique réseau est alors utilisée entre les différentes unités de calculs. De plus, des composantes logicielles sont généralement ajoutées afin de pouvoir synchroniser les données lors, par exemple, de calculs parallélisés (*cluster* ou grappe de PCs). L'utilisation de *Playstation 3* comme moyen de calcul dans un *cluster* est par exemple illustrée dans [BLK+07].

2.6 Les sciences cognitives

Le but principal des sciences cognitives est de s'intéresser aux modes d'interprétation de l'information par l'être humain. Ces sciences font donc appel à un ensemble de disciplines comme la linguistique, l'anthropologie, la psychologie, les neurosciences ou la philosophie et vont permettre d'aider à la compréhension des différents mécanismes qui régissent notre perception. On se rend alors facilement compte à quel point ce domaine est important en réalité virtuelle, car il va permettre de mieux comprendre notre mode de

fonctionnement permettant ainsi de trouver de meilleures solutions pour leurrer nos sens.

Une meilleure connaissance de notre perception permet également de mieux adapter l'utilisation d'un système de réalité virtuelle. Il est alors possible de combiner les sciences cognitives à l'ergonomie pour pouvoir engendrer des applications qui s'adaptent de façon plus naturelle à nos besoins. Il devient possible d'obtenir des solutions qui concilient simplicité d'utilisation, efficacité et convivialité. Outre l'application sur la partie logicielle, l'alliance de l'ergonomie aux sciences cognitives permet de créer des interfaces destinées, au mieux, à produire ou à conserver le sentiment de présence.

2.7 Bilan

À travers la présentation de cet éventail de disciplines, nous avons montré que la création d'une installation de réalité virtuelle demandait des compétences variées. Une équipe de recherche dans ce domaine doit donc faire appel à différents profils afin de s'assurer une pluridisciplinarité de ses compétences.

En ce qui concerne ce mémoire, nous allons nous concentrer plus particulièrement sur l'image, qui d'une part fait partie de notre spécialité et d'autre part est un point que nous considérons comme important en réalité virtuelle. Plus spécifiquement, nous exploiterons différentes méthodes issues de la synthèse d'images et de la vision par ordinateur.

3

État de l'art de la réalité virtuelle

La réalité virtuelle a nécessité l'élaboration de nombreux instruments visant à produire le sentiment de présence. Depuis les balbutiements de la RV, ces interfaces n'ont cessé d'évoluer en devenant de plus en plus sophistiquées ou alors en s'aventurant sur des voies un peu plus exotiques. Nous commencerons par faire un survol de l'historique de la réalité virtuelle pour terminer par une revue des différentes interfaces qui sont actuellement employées.

3.1 Historique

Faire le détail de toutes les évolutions de la réalité virtuelle depuis son apparition est un travail relativement fastidieux. Au gré des recherches qui ont été effectuées et des besoins, bon nombre de systèmes ont vu le jour, mais ce sont les plus innovateurs d'un point de vue scientifique ou applicatif qui restent inscrits parmi les étapes de l'évolution de la réalité virtuelle. Cet historique se limite donc à décrire les points clés qui ont marqué la réalité virtuelle.

Le premier système pouvant être considéré comme destiné à générer une expérience de réalité virtuelle date du début des années 60 et fut pensé et développé par le cinématographe Morton Heilig. Il proposa une installation dont l'objectif était d'ajouter un aspect multi-sensoriel aux films traditionnels et la nomma, en conséquence, «*Sensorama*» [Hei62]. Une de ces fameuses expériences consistait à faire une ballade virtuelle en bicyclette dans les rues de Brooklyn. L'immersion était garantie par la diffusion d'un film stéréoscopique, la vibration du fauteuil, un son stéréo avec, ajouté à cela, la simulation du déplacement grâce à des ventilateurs et l'émission de différentes odeurs. Malgré toutes ces nouveautés, cette interface visible sur la figure 1 ne reçut pas le soutien financier escompté et fût donc par la suite abandonnée.

Quelques années après, Ivan Sutherland décrit dans le papier [Sut65] un nouveau système d'affichage type HMD [Sut68] (*Head Mounted Display*) se basant sur les travaux de Heilig et dont le contenu graphique était aux prémices de ce que l'informatique graphique

serait par la suite. À cela était adjoint un module de *tracking* qui permettait de connaître la position de l'utilisateur à n'importe quel instant et offrait ainsi une certaine interactivité. Avec le soutien de l'armée et de la *NASA* (*National Aeronautics and Space Agency*) qui envisageaient de multiples applications liées à la simulation, il concrétisa son projet et lui donna le nom de «*Sword of Damocles*» en référence à l'aspect visuel que prenait l'installation.

FIG. 1: Une vue du *Sensorama Copyright : http//www.mortonheilig.com/*

En 1970, l'artiste en multimédia Myron Krueger a créé une installation interactive nommée «*Videoplace*» [KGH85] pour illustrer son concept de «*artificial reality*». Ce système se compose d'un écran et d'une caméra dont l'association permet à un utilisateur de pouvoir interagir avec différents environnements virtuels en 2D. Ces derniers sont innovants surtout à cause du fait qu'ils sont capables d'interpréter et réagir aux gestes de l'utilisateur. De plus, avec ce système, Krueger démontre qu'une interaction ne se réalise pas forcément par le biais d'interfaces que doit manipuler l'utilisateur mais peut se faire naturellement, uniquement avec son corps. Cette philosophie a, en plus, l'avantage de libérer l'utilisateur d'une contrainte matérielle qui pourrait, à cause de son poids ou de son encombrement, venir rompre le sentiment de présence. Enfin avec «*Videoplace*», apparaissent les premiers essais de téléprésence car il est possible, pour deux personnes distantes, d'utiliser cette installation via l'affichage de leurs silhouettes respectives à l'écran.

Une équipe de la *NASA* entame en 1981 la création d'un prototype de *HMD* constitué de minis écrans *LCD* qu'elle appelle *VIVED* (VIrtual Visual Environment Display) [FMHR87]. Le but du projet est de développer de nouvelles interfaces pouvant

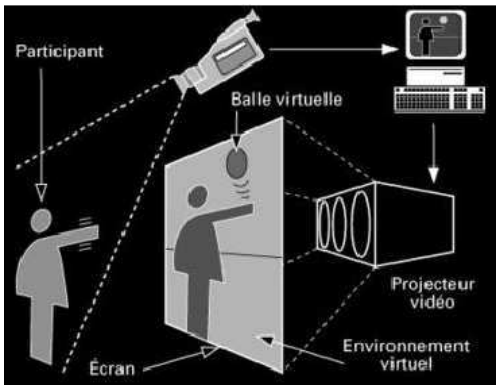


FIG. 2: Principe de l'installation interactive *Videoplace Copyright : Myron Krueger*

permettre l'amélioration des simulateurs destinés aux astronautes. En 1985, le système immersif de la *NASA* est complété par un gant de données qui est fourni par la société VPL qui fut créée par J. Lanier et T. Zimmerman et la première à commercialiser du matériel de réalité virtuelle. Le projet *VIVED* a continué son évolution (ajout d'un *tracking* via une interface Polhemus, possibilité de spatialiser jusqu'à quatre sources sonores en 1988...) jusqu'à à la fin des années 80 où il fut réactualisé et changea de nom pour celui de *VIEW* (*Virtual Interface Environment Workstation*) [Fis99].

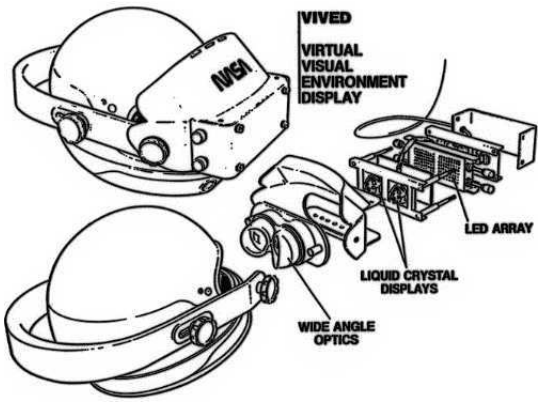


FIG. 3: Description du *HMD* du projet *VIVED Copyright : NASA*

La plus grosse contrainte liée au *HMD* est, dans ces années-là, la combinaison de leur poids et de leur encombrement qui gêne l'utilisateur lorsqu'il veut effectuer des manipulations en réalité virtuelle. Dans l'optique de proposer une nouvelle interface immersive moins envahissante pour celui qui l'emploie, Cruz-Neira *et al.* créent en 1991 un dispositif ayant l'apparence d'un cube et nommé *CAVE* (*Cave Audio Visual Experience*) [CNSD+92]. Le principe est de placer l'utilisateur au centre d'un cube dont certaines faces sont des écrans rétro-projetables, notamment un sur le sol, sur lesquels des images stéréoscopiques sont diffusées. Pour la spatialisation du son, plusieurs enceintes ont été reparties autour de l'installation. Dans les évolutions qui ont suivi le *CAVE* a bénéficié

d'un affichage sur chacune des faces du cube ainsi que d'un système de *tracking* magnétique qui contraint à l'utilisation d'une structure non métallique. Une variante, appelée SAS^{CUBE} [ZA02], a été développée sur cette base quelques années après, mais dans le but de proposer un système qui soit plus facilement transportable pour réduire, entre autres, les coûts d'exploitation.

FIG. 4: Le SAS^{CUBE}

En 1992 est organisée, en France, la première conférence internationale dédiée à la Réalité virtuelle (*Informatique 92 : Interface from real and virtual worlds*) et permet à la communauté scientifique un échange sur les derniers développements technologiques. L'année suivante le *Virtual Reality Annual International Symposium* est organisé par l'*IEEE* à Seattle et à partir de là finit d'asseoir complètement la réalité virtuelle dans le domaine scientifique.

3.2 Les interfaces visuelles de réalité virtuelle

Les interfaces dédiées ou tout simplement employées en réalité virtuelle sont nombreuses et très variées. Dans cette partie, nous allons donc nous restreindre aux seules interfaces visuelles, mais un aperçu des autres interfaces est disponible dans l'annexe A. Nous décrirons dans un premier temps les catégories d'interfaces utilisées en RV et nous présenterons ensuite les procédés employés pour générer des images stéréoscopiques

3.2.1 Les interfaces visuelles

Il est assez difficile d'imaginer un système de réalité virtuelle sans un support visuel, même si cela est envisageable. Chez l'être humain le sens de la vue, en plus d'être l'un des plus compliqués, est l'un des plus utilisés. La principale raison est que c'est par ce sens que l'on peut récupérer un maximum d'informations concernant notre environnement [Hug95, Cut97] et tirer notre plus grande part d'expériences. C'est sur ce principe que s'est fondé notre culture [Cla93]. Les interfaces visuelles peuvent se décrire selon trois

catégories [QRMTHM06] que nous allons détailler.

Les écrans

Les écrans sont historiquement les interfaces visuelles les plus simples pouvant être utilisés en réalité virtuelle mais n'offrant en conséquence qu'une faible immersion en raison de leur taille réduite et induisant généralement une position passive. L'adjonction de multiples écrans permet toutefois de limiter ce problème et contribue à créer un environnement plus immersif comme celui proposé par la société *Panoram Technologies* [ICP04]. Toujours dans cette optique, la société *Alienware* a présenté lors du salon CES 2008 un prototype d'écran incurvé de 48 pouces utilisant une technologie de projection DLP.

Fig. 5: Ecran incurvé *Copyright : Alienware*

Une partie de la recherche sur la réalité virtuelle sur écran a conduit au développement d'un système appelé le *Fish Tank VR* [WAB93, ML00] qui n'ajoute que la prise en compte du point de vue de l'utilisateur pour afficher correctement l'environnement virtuel.

Avec la démocratisation des écrans LCD, le développement des écrans autostéréoscopiques connaît actuellement une croissance importante. Les technologies [Dod05] employées par ces derniers permettent d'afficher des images stéréoscopiques selon différents angles sans avoir besoin de requérir à un quelconque matériel supplémentaire tels que des lunettes ou un détecteur de position. Pour le moment les écrans commercialisés, comme par la firme Phillips [Phi06], permettent un visionnage depuis neuf points de vue distincts. Mais des recherches récentes ont permis d'en obtenir jusqu'à 64 [Tak06].

Les HMDs

Un *Head Mounted display* ou visiocasque (voir la figure 6) est l'une des images les plus représentatives des interfaces utilisées en réalité virtuelle sans doute à cause du côté futuriste qu'elles inspirent. Ce type d'interface est constitué d'un support sur lequel viennent se greffer un ou deux écrans LCD ou cathodiques, selon qu'on veuille une restitution stéréoscopique, ainsi qu'une paire d'écouteurs pour une restitution «spatialisée» du son. Enfin il est possible de coupler cet ensemble avec un *tracking* magnétique afin de connaître à tout moment la position et la direction de la tête de l'utilisateur.

Le premier avantage d'un HMD est sa transportabilité qui lui permet d'être utilisable dans de multiples contextes tels qu'en laboratoire ou alors sur le terrain. La miniaturisation des différents éléments qui le composent a permis d'obtenir un système plus léger et plus flexible que celui que l'on pouvait trouver il y a quelques années (batterie, écrans, câblage). Le deuxième avantage est qu'une interface de ce genre induit une très bonne immersion visuelle et sonore. L'utilisateur est par exemple contraint à ne regarder que l'environnement virtuel et oublie par conséquent le contexte réel. Malgré cela, un HMD souffre de la contrainte majeure de devoir être porté ce qui a pour effet de provoquer une certaine gêne lors de son utilisation et peut alors perturber le sentiment de présence. Une autre contrainte est causée par le côté trop «immersif» qui déroute très rapidement les utilisateurs novices. La rupture brutale avec le monde réel implique un temps d'adaptation qui doit permettre de retrouver des points de repère, mais qui peut être une source de nausées ou de maux de tête. Enfin la proximité des écrans peut entraîner assez rapidement une fatigue oculaire ainsi qu'un champ de vision relativement faible ($\leq 140°$) en comparaison de celui de l'être humain ($\geq 180°$).

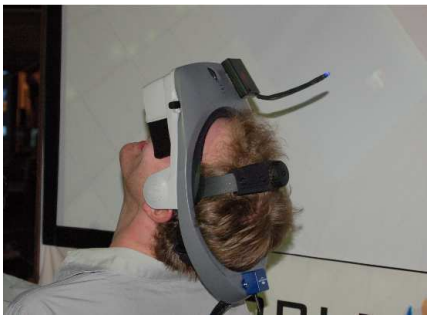


FIG. 6: Exemple d'utilisation d'un *HMD*

La projection d'images

La majeure partie des systèmes de réalité virtuelle utilise une interface visuelle basée sur une projection type *CAVE* [CNSD+92] car c'est celle qui présente un bon compromis entre la polyvalence et l'immersion. On parle dans ce cas de salle immersive étant donné que leurs dimensions généralement assez importantes nécessitent un espace dédié. Ces systèmes sont constitués d'un ou plusieurs écrans (rétro-projetable ou non) de grande taille

dont l'objectif est de remplir au maximum le champ visuel de l'utilisateur et de vidéo-projecteurs adaptés au procédé de restitution stéréoscopique désiré (système actif ou passif). Dans le cas d'une surface de projection relativement grande, plusieurs sources vidéos peuvent être utilisées, dont chacune projette une partie de l'image originale [LPG08]. On trouve par exemple ce type de projection dans le système *PowerWall* [SFF+00]. Ce principe est illustré dans la figure 7.

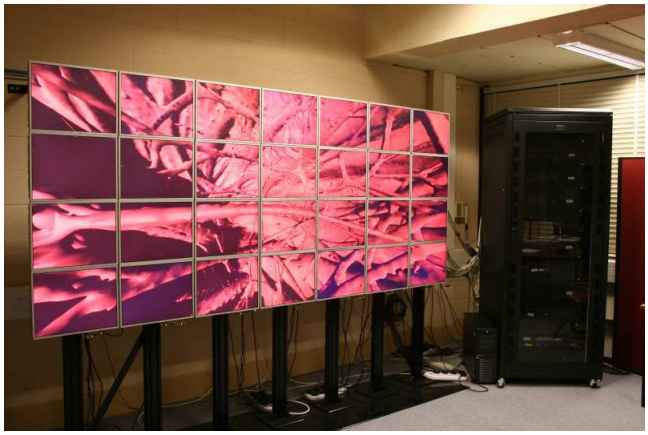


FIG. 7: Exemple d'un *Powerwall Copyright : Université de Leeds*

Parmi les autres systèmes de réalité virtuelle basés sur la projection, nous pouvons citer les bureaux immersifs qui sont en fin de compte assez similaires aux systèmes mentionnés précédemment, mais avec des dimensions moindres et offrant ainsi l'avantage d'être utilisables quasiment partout. On trouve ainsi des installations ayant l'apparence de table dont l'usage est plutôt orienté vers le travail collaboratif, tel que le *WorkBench* [ABM+97], les bureaux généralement constitués de deux écrans [Bar99] pour plus d'immersion et les bureaux sphériques [TWHH02, SHH+03] qui ont l'avantage de mettre l'utilisateur au centre de l'image (l'interaction est toutefois limitée à cause d'une projection depuis le centre de la sphère). Malgré cela, il est intéressant de constater que ces systèmes tendent à rapidement disparaître au profit des salles immersives (tendance qui s'illustre par la fermeture de sociétés telles que *TAN* ou *Elumens* qui étaient spécialisées dans ce type d'interfaces).

Enfin il existe un prototype de système de projection autostéréoscopique [JMY+07] qui se base sur l'affichage à très haute fréquence de 60 images (soit 1440 images par seconde) sur un miroir anisotropique en rotation. L'ajout d'une détection de la position de la tête permet d'ajouter une dimension verticale à la visualisation en trois dimensions de l'objet.

3.2.2 Les procédés de restitution stéréoscopiques

Le binoculaire

C'est incontestablement le procédé de restitution stéréoscopique le plus ancien qui soit connu car il utilise la méthode la plus simple : faire en sorte que chacun de nos yeux

observe une image différente. De manière minimaliste il peut s'agir de deux images présentant une même scène mais depuis un point de vue légèrement décalé. Une fusion des deux images est alors nécessaire pour créer l'effet de relief et cela n'est possible qu'en appliquant une vision croisée ou une vision parallèle. Un exemple de telles images est présenté figure 8.

(a) (b)

FIG. 8: L'image (a) illustre le procédé de restitution stéréoscopique par vision croisée, tandis que l'image (b) illustre celui de la vision parallèle

Afin de simplifier la fusion des deux images, cette technique a été adaptée sur une monture, le stéréoscope, qui peut être assimilé à une paire de jumelles : chaque oeilleton ne permet de voir qu'une seule image. Ainsi la reconstruction du relief se fait presque naturellement car elle est très similaire à notre mode de vision. Un exemple de cet appareil est présenté sur la figure 9.

FIG. 9: Le stéréoscope

L'évolution logique a été de remplacer les images fixes par des images animées et s'est concrétisée par l'apparition des «Head Mounted Displays» comme décrit dans la partie 3.2.1.

Le filtrage

Pour associer une image à chaque œil, il existe d'autres techniques de restitution qui sont plus adaptées à une projection sur écran. En 1853, un américain du nom de Rollman fait la découverte d'un procédé de filtrage qui est amélioré un demi siècle plus tard par Charles Ducos qui lui donnera le nom d'anaglyphe. Le principe se fonde sur le filtrage des deux images nécessaires à la stéréoscopie dont une se fait via un filtre rouge et l'autre

par un filtre cyan. Il est alors possible de les fusionner en une seule image sans perte d'information au niveau du contenu visuel. La reconstruction du relief se fait en portant une paire de lunettes dont les verres ont été remplacés par les mêmes filtres utilisés pour créer l'anaglyphe. Une paire de lunettes et une image anaglyphe sont illustrées dans la figure 10. Malgré tout, ce procédé a le désavantage d'altérer les couleurs ; pire encore, certains objets de l'image qui sont rouges ou cyans ne peuvent être vus que par un seul œil, dégradant ainsi le relief. Toutefois, cette technique reste la plus courante car elle est très simple et très peu coûteuse. Il n'est d'ailleurs pas rare de visionner des films au cinéma utilisant l'anaglyphe.

FIG. 10: L'anaglyphe

Il existe un autre système de stéréoscopie assez similaire à l'anaglyphe appelé *Color Code 3D* [3-D] mais qui est encore sous licence. Son avantage est de limiter la perte d'informations chromatiques en proposant des lunettes se composant d'un filtre ambre et d'un filtre bleu. Le premier va permettre de restituer pratiquement toutes les plages de couleurs (des limites existent avec le jaune et le bleu) tandis que le second ne va transmettre que l'information de parallaxe (image désaturée sur laquelle un filtre passe-haut a été utilisé).

Un autre procédé basé sur le filtrage a été mis au point en 1933 par Bernard Lyot. Ce filtre permet de sélectionner certaines ondes de la lumière en se basant sur leur nature éléctomagnétique. Par défaut, une onde émise par une source de lumière n'a aucune orientation par rapport à son axe de diffusion. En plaçant un filtre polarisant sur sa trajectoire, il est possible de ne transmettre que les ondes qui ont une orientation bien précise. Dans un système stéréoscopique, deux images sont projetées via des vidéo-projecteurs devant lesquels des filtres polarisant la lumière ont été placés selon deux directions opposées. L'utilisateur n'a plus qu'à porter une paire de lunettes (figure 11) avec les mêmes filtres pour reconstruire le relief. Il est toutefois à noter que la surface de projection doit aussi être polarisée afin que l'orientation des ondes soit conservée. Enfin, le filtrage peut se faire de deux manières : soit par une polarisation linéaire (vertical/horizontal), soit par une polarisation circulaire (sens trigonométrique/sens inverse). L'avantage de ce type de filtrage sur l'anaglyphe est qu'il n'y a pas d'altération des couleurs, mais une forte diminution de l'intensité lumineuse.

Une dernière technique de filtrage, proposée par Jorke et Fritz [JF03], se nomme *Infitec*. Elle repose sur la segmentation du spectre de la lumière visible en deux. Ainsi, via un dispositif de filtrage, chaque œil va recevoir une moitié des longueurs d'onde de chacunes des composantes rouge, verte et bleue du spectre (figure 12).

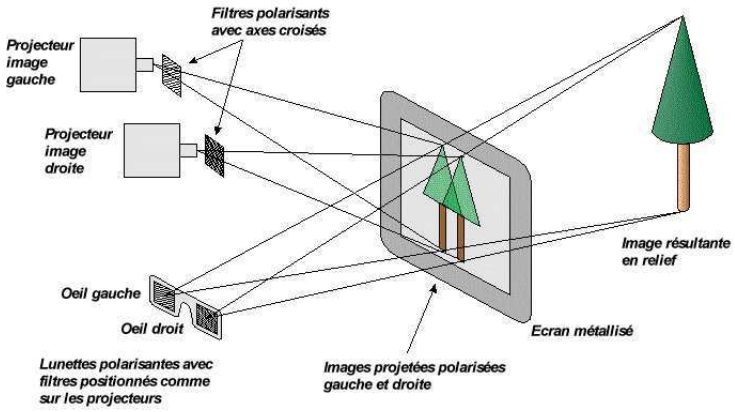


FIG. 11: Le procédé de stéréoscopie utilisant la polarisation de la lumière *Copyright : stereoscopie.org*

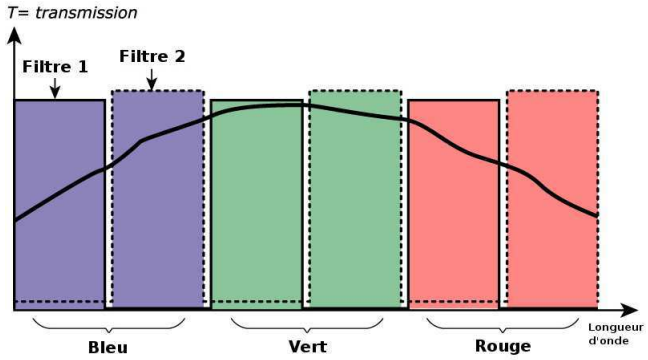


FIG. 12: Le procédé de stéréoscopie *Infitec Copyright : Barco*

L'obturation

Le premier système, dit à obturation, est mis au point en 1922 par Laurens Hammond et William F. Cassidy et fondé sur les travaux de Charles d'Almeida. Le principe de cette méthode repose sur la diffusion d'une série d'images dont une sur deux est destinée à l'œil droit et l'autre à l'œil gauche. Pour les séparer l'observateur doit porter une paire de lunettes synchronisées avec l'affichage et qui va alors alternativement obturer la lunette gauche ou la lunette droite. Les systèmes actuels sont basés sur des montures à cristaux liquides qui sont synchronisés avec l'affichage via une transmission infrarouge 13. Cela signifie que les lunettes doivent embarquer une alimentation pour permettre à l'obturation de fonctionner ce qui a pour conséquence de fortement les alourdir (et provoquer par exemple un mal de nez) ou peut cesser d'être alimenté en cours d'utilisation. De plus un léger décalage dans la synchronisation peut rendre l'expérience très désagréable.

Il est désormais possible dans certaines salles de cinéma numérique de pouvoir assister à une projection en relief utilisant l'obturation, qui malgré une légère baisse de la luminosité des images, permet de restituer pleinement les couleurs. Ce dispositif était, par

FIG. 13: Exemple de paire de lunettes utilisant le procédé d'obturation

exemple, employé pour le film «Voyage au centre de la terre 3D».

Les autres techniques

L'effet *Pulfrich* découvert par le physicien du même nom en 1922 décrit un phénomène de décalage temporel de réception entre les deux yeux. Le phénomène peut notamment se produire lorsqu'un filtre sombre est placé devant l'œil droit. Dans le domaine de la stéréoscopie, l'effet *Pulfrich* peut être utilisé pour percevoir le relief sur des *travelling* vidéo. De cette manière chaque œil reçoit l'image d'une même scène mais avec un point de vue légèrement décalé. Il était par exemple possible de voir ce type de relief dans le dessin animé «*The bots master*».

Le procédé appelé *chromadepth*, inventé par Richard Steenblik en 1983, se base sur la diffraction de la lumière pour créer l'effet de relief. Deux prismes fins, placés sur une paire de lunettes, vont séparer le spectre lumineux pour envoyer les couleurs sur différentes positions de la rétine. Ainsi la couleur rouge va sembler être en premier plan tandis que la couleur bleue sera dans l'arrière plan. Cela signifie donc que l'image de départ doit être encodée avec les bonnes couleurs en fonction des effets de profondeur que l'on souhaite obtenir. Cette technique est, par exemple, utilisée pour la visualisation de données topographiques qui ne nécessitent pas toujours une conservation des couleurs d'origine (figure 14).

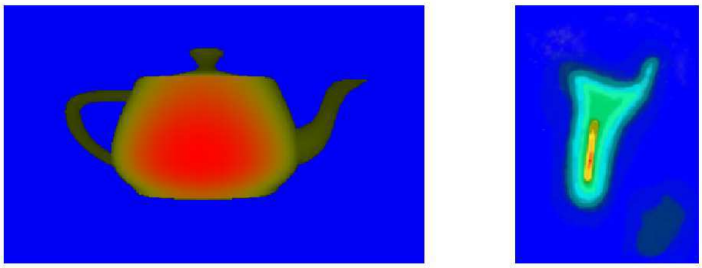


FIG. 14: Deux exemples d'images dont le relief peut être perçu grâce au procédé *chromadepth*

Les auto-stéréogrammes ou SIRDS (*Single Image Random Dot Stereogram*) sont issus d'une technique de stéréoscopie découverte par le Docteur Bela Julesz et perfectionné par Christopher Tyler en 1979. Ils ont la particularité de ne nécessiter qu'une carte de profondeur. Pour chaque pixel d'une ligne, on récupère la profondeur correspondante pour en déduire les projetés pour l'œil droit et l'œil gauche. Si ces deux positions sont dans l'espace de visualisation alors on leur attribue une même couleur aléatoire ou issue d'un motif. Pour percevoir l'effet de relief sur l'image finale, l'observateur doit loucher en s'aidant de deux points noirs généralement placés en haut de l'image, comme sur la figure 15. Petz *et al.* [PGM03] proposent une implantation de stéréogramme temps réel utilisant le GPU.

FIG. 15: Exemple de stéréogramme (la réponse est une thèiére)

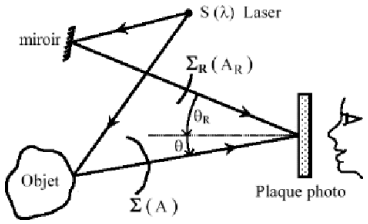


FIG. 16: Principe de création d'un hologramme. *Copyright : Wikipédia*

Le principe de l'holographie ou photographie 3D, découvert par Dennis Gabor en 1948 et développé par Leith et Upatneik en 1960, consiste à enregistrer la phase (distance) en plus de l'amplitude (intensité) de la lumière provenant d'un objet. Une plaque photographique ne pouvant basiquement permettre cela, l'interférence de deux faisceaux de lumière cohérents (LASER) est utilisée. Le premier faisceau, dit de référence, est directement projeté sur la plaque et le second, dit objet, est d'abord envoyé sur le sujet que l'on souhaite enregistrer, puis est réfléchi en direction de la plaque (figure 16). La restitution se fait en éclairant la plaque développée par un LASER. Dans certains cas,

il est possible de visualiser un hologramme en utilisant une source de lumière blanche à condition que la profondeur des objets photographiés ne soit pas trop grande.

4

Domaines d'application de la réalité virtuelle

La réalité virtuelle est un domaine qui, malgré les apparences, n'est pas restreint aux laboratoires. Les faits sont, en effet, bien différents même si l'usage de la réalité virtuelle, à proprement parlé, reste réservée à un petit nombre de personnes. Nous allons voir dans cette partie les différents domaines qui exploitent la RV et ce qu'ils en retirent.

4.1 La simulation

Lorsque l'on aborde le sujet des domaines d'applications de la réalité virtuelle, le premier à venir à l'esprit est la simulation. La principale raison de cette association est l'image véhiculée par les médias ainsi que la plus-value technologique qui profite aux industriels en présentant un tel système. Outre cela, il y a un réel intérêt à employer la RV dans un secteur comme la simulation.

Le premier concerne l'aspect financier. De nombreuses grandes entreprises ont adopté un système de réalité virtuelle qui va être dédié à l'étude d'un de leur produit. On trouve généralement les constructeurs automobiles, aéronautiques ou les secteurs militaires. L'avantage principal est la possibilité de pouvoir tester un appareillage en considérant divers paramètres qui peuvent varier d'une simulation à l'autre. Le bénéfice est clairement la modularité qu'apporte le système de réalité virtuelle couplé au gain de temps d'exécution d'une simulation par rapport à une simulation effectuée sur le terrain.

L'autre apport de la réalité virtuelle dans la simulation est visible dans le cas d'une formation d'un personnel à une activité qui nécessite une phase d'apprentissage. Le mot clé associé à ce type de simulateur est «sans danger» car les utilisateurs ne risquent à aucun moment d'être victime d'un accident. L'apprentissage se fait donc en toute sécurité tout en étant généralement accompagné d'une équipe d'encadrement qui peut facilement contrôler et analyser les différentes actions faites par l'utilisateur comme sur la figure 17. Un exemple relativement connu de système de formation est le simulateur d'avion qui permet aux pilotes, débutants comme expérimentés, de pratiquer leur activité dans des

situations peu courantes et délicates mais avec une garantie de sécurité. Ce genre de choses est absolument impossible hors de son contexte. Utiliser la réalité virtuelle pour une formation offre ainsi la possibilité de mettre les utilisateurs dans une situation virtuellement dangereuse, mais sans aucun danger réel.

FIG. 17: Exemple de plate-forme de simulation avec l'espace d'analyse à l'arrière

4.2 Les activités ludiques

Un emploi remarquable de la réalité virtuelle est son utilisation dans le cadre d'activités ludiques et plus particulièrement, celui du jeu vidéo. C'est principalement dans ce contexte que le tout à chacun a la possibilité de pouvoir expérimenter une installation de réalité virtuelle. Cette dernière se prête bien à ce contexte car l'objectif d'un jeu vidéo est d'incarner un personnage qui évolue dans un univers virtuel. Autrement dit, le jeu vidéo présente les conditions nécessaires pour générer le sentiment de présence chez son utilisateur.

Il est courant de rencontrer cette association de la réalité virtuelle avec le jeu vidéo dans les salles d'arcade qui présentent un large éventail d'installations [Bou]. Il est par exemple possible de prendre place dans un simulateur de voiture de courses (*Daytona USA 2*), utiliser une arme factice pour détruire des ennemis (*The House of the Dead 4*), monter un cheval lors d'une course (*Jockey Club*), chausser des skis pour dévaler une pente (*Alpine racer*)... En 1998, le groupe *Disney* a créé un parc d'attraction nommé *DisneyQuest* qui est entièrement dédié à la réalité virtuelle avec des activités, comme le *Aladdin's Magic Carpet Ride* qui fait vivre un voyage virtuel sur tapis volant à l'aide d'un *HMD*.

Le principal inconvénient de cette utilisation de la réalité virtuelle est qu'elle reste assez occasionnelle, principalement parce que de telles installations ne sont pas utilisables

dans les foyers. Pour en bénéficier il faut donc se déplacer vers des lieux qui leur sont dédiés. Toutefois, depuis peu il est possible de constater une émergence d'un nouveau genre de consoles de salon qui mettent l'accent sur l'interaction. Avec la *Wii* de *Nintendo*, il devient par exemple possible de jouer au tennis en reproduisant les gestes de frappe de la balle tout en restant devant sa télévision.

4.3 Les activités artistiques et culturelles

Dans le cadre de nombreuses expositions artistiques, il est possible de découvrir des installations interactives. Ces dernières font assez souvent appel à des principes similaires à ceux de la réalité virtuelle. Le précurseur dans ce domaine était notamment Myron Krueger qui avait créé une installation nommée *Vidéoplace*. D'une manière générale le principe d'une installation interactive est de faire partager une idée artistique en utilisant des technologies permettant de faire passer le visiteur de simple spectateur à un statut d'acteur. Un exemple de médiatisation des installations interactives est l'organisation des «nuits blanches» tous les automnes dans les rues de Paris.

La réalité virtuelle est aussi utilisée pour faire des bonds dans le temps. Dans le cadre de recherches sur les anciennes cultures de nombreuses reconstitutions ont vu le jour. Recréer virtuellement une antique cité apporte un autre point de vue permettant de vérifier différentes théories. Les historiens ont alors la possibilité de se projeter directement dans un «Pompéi» virtuel tel que la ville devait être lors de son apogée et dans laquelle ils vont pouvoir déambuler et mieux se rendre compte si leurs hypothèses étaient les bonnes [FPR+01].

4.4 Le domaine médical

La réalité virtuelle a une triple utilité dans le domaine médical et plus particulièrement en chirurgie. On peut facilement imaginer à quel point la formation d'un chirurgien peut être délicate. La première mise en pratique d'un certain nombre d'acquis n'est pas forcement simple à cause du stress ou de l'imprécision du geste. La formation passe généralement par un entraînement sur des animaux, mais qui ne reflète pas toujours la réalité ou les complications inattendues. La réalité virtuelle offre ainsi la possibilité de pouvoir s'entraîner en s'aidant d'interfaces haptiques dédiées.

L'autre aspect de la chirurgie concerne la spécialisation du personnel. Dans certains cas une opération peut nécessiter une intervention particulière qui ne peut être effectuée que par un nombre limité de spécialistes dans le monde. C'est pour ces raisons que la télé-chirurgie a vu le jour. En septembre 2001, a eu lieu l'«opération Lindbergh» qui est le premier acte de ce genre sur un patient. L'opération réalisée à Strasbourg et conduite depuis New-York (soit 15000km) nécessitait une technologie de précision via un bras robo-

tisé avec une connexion très haut débit. Cette utilisation de la réalité virtuelle préfigure l'avenir des actes chirurgicaux.

Enfin, la dernière application de la réalité virtuelle au domaine médical concerne les thérapies. La mise en condition du patient est une étape très importante dans la recherche d'une solution à son problème ou en tant que «soin». On peut tout d'abord distinguer le traitement des phobies comme la claustrophobie ou l'agoraphobie [BBP+98, NNC98, MKMA02]. La réalité virtuelle propose alors au patient de se confronter à ses peurs, en portant un HMD permettant une mise en situation. Le second aspect est d'aider les individus ayant subis un événement post-traumatique. Les victimes de guerres ou d'attentats ont généralement besoin de l'assistance d'un psychiatre pour mieux accepter les événements passés et tenter un retour normal au quotidien. La réalité virtuelle est dans ce cas un soutien important car elle permet de faire revivre ces événements de manière contrôlée et complètement analysée pour trouver une solution au problème psychologique [RPG+06, DH02].

Deuxième partie

Outils d'aide à la perception visuelle en réalité virtuelle

1

Définition du besoin

Parmi les quatre piliers de la réalité virtuelle, que nous avons mentionné dans la première partie, nous allons nous intéresser plus particulièrement à celui de l'immersion. Son objectif consiste à tromper nos sens, afin de nous faire percevoir un élément qui est censé être purement virtuel. Comme cela a été précédemment évoqué, la vue est le sens que nous utilisons culturellement le plus ce qui explique qu'il soit à la base de la plupart des systèmes de réalité virtuelle (RV).

Dans cette partie nous allons proposer différents outils destinés à aider, voire accroître l'immersion visuelle. Après nous être intéressé au contexte dans lequel se positionnent ces travaux, nous présenterons une première méthode destinée à accélérer le rendu stéréoscopique ainsi qu'autostéréoscopique (multi-points de vue). Enfin nous étudierons comment il est possible d'aider un observateur à focaliser son attention sur des éléments donnés d'une scène en utilisant le flou de profondeur de champ. Nous verrons également que cette méthode permet de limiter les effets de bords induits par l'affichage d'images stéréoscopiques comme les maux de tête.

1.1 Objectifs de l'immersion visuelle

Dans chaque système de réalité virtuelle, les interfaces les plus misent en avant en raison de leur importance dans le processus de perception de l'environnement sont celles liées à la stimulation visuelle. Il est possible de cataloguer les améliorations apportées au cours des dernières années à ce type d'interfaces en deux parties. D'une part, nous allons trouver les travaux de recherche portant sur les supports d'affichage, avec l'apparition de surfaces de projection complètement immersives comme le *Cave*, celles plus facilement transportables comme les *HMD*, ou affichant des images stéréoscopiques sans avoir besoin de porter une quelconque interface. D'autre part, nous trouvons l'ensemble des méthodes qui permettent de constamment accroître le réalisme et la complexité des environnements virtuels qui tendent de plus en plus vers le photo-réalisme et qui s'affichent en temps réel.

En réalité virtuelle, les moyens mis en œuvre pour accroître l'immersion sont, pour la

plupart, fondés sur une solution matérielle. Actuellement la tendance de la recherche se focalise sur l'utilisation de systèmes de moins en moins onéreux ce qui entre généralement en conflit avec le matériel proposé sur le marché ou les créations d'interfaces expérimentales. La recherche en réalité virtuelle se tourne donc vers des solutions algorithmiques qui peuvent également permettre d'accroître l'immersion.

Pour ces raisons, il est possible d'améliorer l'immersion visuelle en proposant des solutions algorithmiques qui ont l'avantage de ne pas nécessiter d'interfaces hors de prix. L'intérêt de se concentrer sur la vue, est que c'est le sens le plus utilisé et dont l'utilisateur est le plus sensible. Le rôle de la vue dans le sentiment de présence a donc une réelle importance, d'où la nécessité d'axer la recherche vers de nouvelles solutions visant à perfectionner les éléments d'immersion visuelle.

1.2 Contexte d'application

La plupart des systèmes de réalité virtuelle «clés en main» proposés sur le marché sont généralement très onéreux (coût supérieur à 150.000€) en raison de la qualité du matériel fournit. De plus, ces salles sont rarement transportables à cause de leurs dimensions importantes et du temps nécessaire à leur mise en place. Ces raisons font que l'accessibilité de la réalité virtuelle est réservée à une faible quantité de personnes ne favorisant pas sa démocratisation.

Dans le cas concret de notre salle de réalité virtuelle, nous avons pris le parti d'un système orienté faible coût et mobilité. Elle peut alors être décrite comme une installation composée d'éléments grand public avec par exemple l'utilisation d'ordinateurs communs et dont la mise en place peut se faire en un temps assez court, de l'ordre d'une demi-journée. Outre la volonté d'utiliser une salle de réalité virtuelle fondée sur ces principes, nous cherchons à créer un sentiment de présence en faisant en sorte que l'utilisateur agisse de manière naturelle. Cela implique que ce dernier doit être en contact avec le moins possible d'interfaces et se résume dans notre cas au port de lunettes polarisantes et de diodes pour le *tracking* des postures.

Notre installation de réalité virtuelle se compose d'un système de projection à polarisation radiale sur un écran de taille $5m \times 2.5m$ pour complètement envahir le champ de vision de l'utilisateur. L'espace où se trouve ce dernier est observé par plusieurs caméras qui permettent de connaître la position de diodes qui sont portées par l'utilisateur. Au sol se trouve un tapis sensitif fondé sur deux feuilles d'aluminium qui entrent en contact lorsqu'une pression s'applique dessus et qui permet ainsi de connaître approximativement la position de l'utilisateur. Autour de cet espace sont réparties huit enceintes qui sont positionnées sur une sphère fictive dont la tête de l'utilisateur est placée au centre. L'interaction avec l'environnement virtuel est assuré soit par la reconnaissance des mouvements effectués par l'utilisateur, soit par des interfaces simples comme une *Wiimote* de la firme *Nintendo*.

2

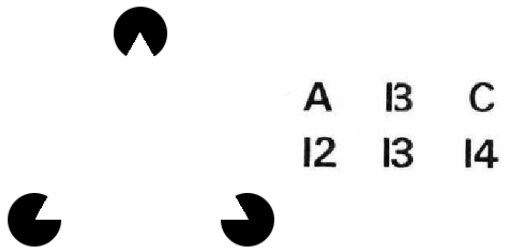
Stéréoscopie et accélération graphique

2.1 La perception du relief

Dans notre société contemporaine, l'information visuelle (photographie, télévision, cinéma, etc.) se transmet majoritairement sur des supports 2D. Les rares exceptions d'images en relief se retrouvent concentrées dans quelques parcs d'attractions tel que *Futuroscope*, dans des films peu populaires comme «*Mission 3D Spy kids 3*» ou dans quelques images en relief. Les facteurs qui caractérisent ce faible succès sont principalement liés aux conditions techniques nécessaires à leur réalisation, à leur diffusion et au mal de crâne que le relief peut provoquer. La création d'une image stéréoscopique, ou par extension un film, nécessite que la capture se fasse depuis deux points de vue légèrement décalés par rapport à l'axe optique. Le développement d'un appareil capable de satisfaire cette contrainte demande beaucoup plus de moyens financiers dus en partie aux réglages minutieux dont il a besoin (positionnement des objectifs, synchronisation des flux,...). Les conditions de prise de vue sont également différentes, car il faut tenir compte des éléments de la scène que l'on désire mettre en profondeur ou bien en relief, tout en veillant à ne pas exagérer les effets, afin d'éviter que l'utilisateur ait trop de difficultés pour reconstruire l'image finale.

Cette dernière raison limite l'utilisation des images stéréoscopiques, car fusionner deux images pour créer le relief, même via une interface visuelle dédiée, peut entraîner de violents maux de têtes en cas d'utilisation prolongée. La principale cause est liée au phénomène d'accommodation-convergence associé à une image en relief : nos yeux vont faire la «mise au point» au niveau de l'écran tout en convergeant vers un point de l'espace différent. C'est cette disparité que notre cerveau a du mal à accepter. De plus, il se peut que certaines personnes n'arrivent pas à faire la reconstruction du relief à cause de petits défauts présents au niveau de l'oeil (strabisme, oeil faible, ...). Enfin, des contraintes surviennent lors de la diffusion, car des supports tels que la télévision ou les cinémas classiques ne sont pas adaptés, ce qui entraîne l'utilisation de procédés de restitution simples comme l'anaglyphe qui ne modifient pas le système de projection mais qui, par contre, modifient le rendu des couleurs (l'utilisation du rouge est par exemple à proscrire).

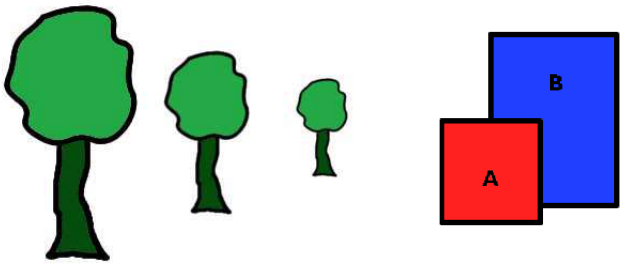
Les images stéréoscopiques ne sont pas l'unique moyen de percevoir l'agencement en profondeur des différents éléments qui composent une scène, car un certain nombre d'autres informations peuvent être exploitées. La perception visuelle se fonde sur deux notions qui sont notre expérience et notre connaissance du monde. Au cours de notre existence nous avons appris à développer notre sens de la perception à travers différentes situations que l'on a pu rencontrer. Son utilité se concrétise, par exemple, avec notre aptitude à compléter ou interpréter des formes incomplètes comme celles présentées sur la figure 18.



(a) (b)

FIG. 18: Deux exemples de l'influence de l'expérience sur la perception visuelle. Sur l'image *(a)* il est possible de distinguer un triangle alors qu'il n'y en a aucun de présent. Sur l'image *(b)* le deuxième symbole de la première série sera interprété comme un B alors qu'il pourrait s'agir du chiffre *13*

La connaissance que nous avons de notre monde, constitué de lois physiques ou de concepts logiques, a un rôle important dans la perception visuelle que nous en avons. Il est possible de l'utiliser pour obtenir des informations sur les relations entre les différents composants qui constituent notre environnement comme leurs distances relatives. Le premier cas de figure concerne la taille des objets, car nous pouvons évaluer la distance à laquelle ils se trouvent si nous connaissons approximativement ses dimensions. La figure 19a illustre ce principe en présentant trois arbres de même aspect mais de tailles variables. Ce que nous en déduirons en premier est qu'ils sont à des distances différentes car nous allons nous référer à l'arbre le plus grand comme dimension étalon. La perception de la profondeur peut être aussi obtenue via les occultations provoquées entre les éléments. Si, sur une image, un objet A semble se superposer à un objet B (figure 19b), la conclusion la plus logique sera de dire que l'objet A est devant le B mais cette configuration pourrait aussi être interprétée comme un objet A adjacent à un objet B (pas de prolongement de B sous A).



(a) (b)

FIG. 19: En l'absence de données supplémentaires, il est possible de donner deux interprétations à chacune de ces images : *(a)* les arbres peuvent être sur un même plan ou à des profondeurs différentes, *(b)* A est devant B ou A est adjacent à B.

Dans les exemples précédents nous avons vu que le manque de données supplémentaires dans les images pouvait conduire à une ambiguïté sur l'interprétation de son contenu, ainsi des informations supplémentaires peuvent rentrer en ligne de compte pour lever les doutes. La première est associée aux ombres (Figure 20) qui permettent de situer mutuellement des objets. Par exemple l'ombre d'un objet peut se projeter sur un autre. De plus, l'illumination d'une surface va mettre en évidence certains de ses détails (rugueuse, lisse, ...), en plus de leur orientation spatiale. L'accumulation de ces indices va aider à la perception visuelle.

FIG. 20: La projection d'une ombre d'un objet sur un autre objet permet de mieux déterminer les distances mais aussi les formes géométriques

FIG. 21: Perspective de surface : plus un objet va être éloigné plus son apparence va subir une atténuation due aux effets atmosphériques

Il existe d'autres manières de savoir si un objet est éloigné ou non par rapport aux autres présents à l'image en utilisant les éléments de perspective. Le premier est la per-

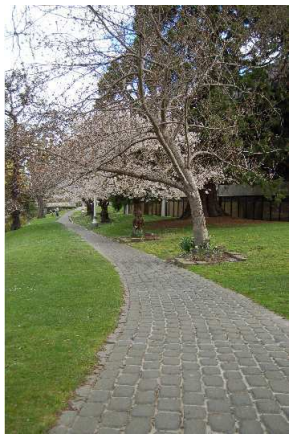


FIG. 22: Perspective linéaire : le tracé du chemin semble se rejoindre en un point, pourtant il est bien perçu comme deux lignes situées à une distance fixe

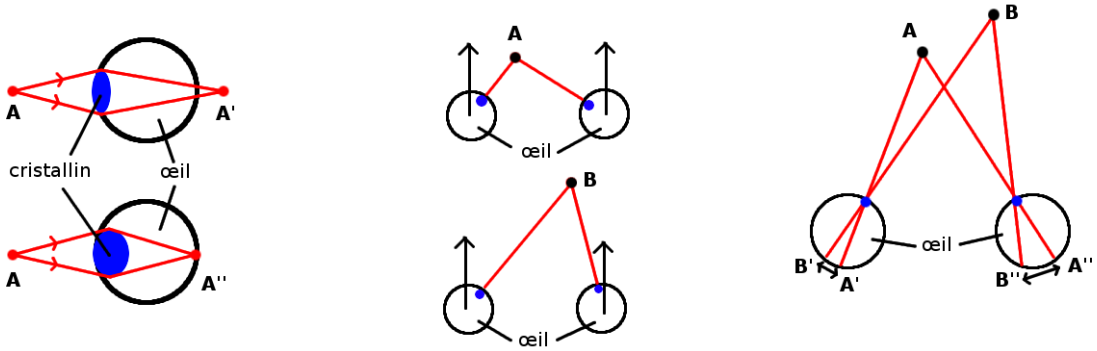
FIG. 23: Perspective de texture : plus un élément est éloigné, plus il sera difficile d'en distinguer les détails de sa texture comme avec les brins d'herbe de la pelouse

spective de surface. Elle est liée à l'atténuation provoquée par l'épaisseur de la couche atmosphérique comme le montre la figure 21 qui va diminuer progressivement la visibilité des objets au fur et à mesure qu'ils sont distants. Il est également possible d'évaluer la profondeur des objets dans une image avec la perspective linéaire. Sur la figure 22 on peut voir les limites d'un chemin qui semblent fusionner au niveau de la «ligne d'horizon». Malgré tout nous percevons cela comme l'éloignement de deux lignes parallèles. Cette information nous permet alors de dire qu'un élément situé sur au bout du chemin est éloigné. C'est cette perspective qui peut par exemple être utilisée pour résoudre l'ambiguïté introduite par les arbres de la figure 19a. Le dernier indice qui peut nous permettre de dire si un objet est éloigné ou non, consiste à utiliser la perspective de texture qui décrit la perception des détails sur une surface. Un exemple est illustré sur la figure 23 qui présente un espace vert : sur une partie de l'image il est possible de distinguer dans une zone des brins d'herbes ou des feuilles mortes, ce qui sera perçu comme proche, par contre la zone où la surface paraît être régulière sera perçue comme éloignée.

Un dernier moyen permettant de percevoir une impression de relief, consiste à exploiter le mouvement des objets entre eux, car la vitesse d'un objet proche va apparaître plus rapide que celle d'un objet éloigné. Ce facteur, appelé parallaxe, peut être exploité sur des écrans monoscopiques en effectuant soit un *tracking* de la tête de l'observateur qui modifie en conséquence le contenu de la scène visionnée. Un dérivé de l'utilisation de la parallaxe, dûe au mouvement, est l'effet Pullfrich qui permet grâce à une paire de lunettes spéciale d'obtenir le relief sur un travelling.

Si nous sommes capables de percevoir la «troisième dimension» à l'aide de tous ces indices, nous ne devrions théoriquement pas avoir besoin de créer des images stéréoscopiques. En fait notre vision est basée d'une part sur les facteurs monoculaires qui

regroupent les éléments de perception que nous venons de citer avec l'accommodation, et d'autre part sur les facteurs binoculaires qui regroupent la convergence et la disparité binoculaire. Regarder une séquence ou une image en deux dimensions utilise ces deux facteurs, mais en sous-exploitant certaines de leurs composantes. Ainsi l'accommodation (figure 24a), qui correspond à l'adaptation du cristallin de l'œil pour effectuer une mise au point, la convergence (figure 24b), qui est la focalisation des axes optiques de nos yeux sur un point de l'espace, et la disparité binoculaire (figure 24c), qui est la différence de distance qui existe entre deux objets projetés sur la rétine, ne sont pas utilisées dans ce cas, car tous les éléments sont sur un même plan.



(a) (b) (c)

FIG. 24: Les trois facteurs qui sont faiblement utilisés sans stéréoscopie : *(a)* l'accommodation, *(b)* la convergence et la *(c)* disparité binoculaire

Ainsi l'utilisation de la stéréoscopie va permettre d'exploiter ces trois facteurs et simuler correctement notre perception du relief. L'objectif va être de générer deux images à partir de deux points de vue légèrement décalés (généralement d'une distance interoculaire moyenne de 6cm) et de les associer à chacun de nos yeux. Il est possible de voir une liste des différentes méthodes de restitution stéréoscopique dans la section 3.2 du précédent chapitre.

2.2 Images stéréoscopiques : État de l'art

En synthèse d'images, le principe de création d'une image stéréoscopique est toujours le même quelle que soit la technique de rendu utilisée. Le détail des calculs est donné dans l'annexe B mais peut se résumer à rendre la scène deux fois de suite depuis deux points de vue légèrement décalés. Cette obligation entraîne par conséquent le quasi-doublement des temps de calculs nécessaires pour chaque rendu, mais qui peut être légèrement réduit en pré-calculant la position des triangles dans la scène, dans le cas d'une animation par exemple.

Une première optimisation est utilisée avec le lancer de rayons qui est un rendu réputé pour être lent, mais réaliste. Lorsqu'il s'agit de créer une image stéréoscopique les temps de calculs deviennent alors très longs. Adelson et al. proposent dans leur méthode [AH93] de commencer par générer la vue de gauche et d'exploiter le fait que certains pixels projetés pour cette vue sont partagés par la vue de droite via un décalage sur l'axe horizontal uniquement. Cette méthode génère différentes contraintes qui sont dues à la manière dont les pixels vont être reprojetés. En imposant un parcours de gauche à droite sur chaque ligne on évite le problème du choix du pixel à afficher lorsque plusieurs sont projetés en un même endroit. Dans certaines situations, deux pixels voisins dans la vue de gauche peuvent ne plus l'être après projection dans la vue de droite. Ce cas, caractéristique des occlusions, peut entraîner un certain nombre d'aberrations dans le résultat. La solution consiste simplement à supprimer les pixels à l'origine du problème. Pour combler les trous, un lancer de rayon est réappliqué sur chacun des pixels concernés. Malgré ces contraintes qui demandent un certain nombre de calculs supplémentaires, l'optimisation permet d'économiser jusqu'à 95% du temps qu'il aurait fallu pour générer la seconde vue avec la méthode traditionnelle. Finalement même si les bénéfices sont significatifs, cette méthode utilisant le lancer de rayon est difficilement applicable dans le contexte d'une application temps réel.

Dans le cadre d'un rendu traditionnel, c'est-à-dire avec la projection et la rastérisation de polygones, il n'existe pas d'algorithme permettant d'accélérer la création d'images de synthèse stéréoscopiques. La société *NVIDIA* a néanmoins ajouté une couche dans les pilotes de ses cartes graphiques [NVI03] pour que n'importe quelle application ayant un contenu 3D calculé en temps réel puisse être affiché en relief. Le rôle du pilote est d'exécuter deux fois les instructions d'affichage d'un programme en appliquant les bonnes transformations pour la vue de droite et la vue de gauche et en stockant le résultat dans deux *buffers* de rendu distincts pour pouvoir être utilisés avec n'importe quel type de procédé de restitution stéréoscopique. Dans le cas de l'utilisation de programmes sur les *shaders* dans l'application, le pilote de *NVIDIA* va les récupérer et les modifier en tenant compte de la profondeur des sommets pour calculer le bon décalage à appliquer. Le principal avantage de cette méthode est qu'elle peut s'appliquer quelle que soit l'API utilisée (*OpenGL* ou *DirectDraw*) et qu'elle n'a pas besoin de l'intervention de l'utilisateur pour s'appliquer (si ce n'est au niveau des réglages des paramètres stéréoscopiques). Toutefois, il n'est possible de l'utiliser que sous un système *Windows* muni d'une carte graphique *NVIDIA* (inférieure à la famille des GeForce 8 sous *Windows XP*). Enfin pour éviter des problèmes d'affichage lors du rendu, des règles rigoureuses de développement au niveau du code et du contenu de la scène doivent être appliquées [NVI06].

Aucune de ces méthodes ne fait réellement usage des possibilités de la carte graphique et plus précisément des *shaders* (détails dans l'annexe C). En fait la programmation sur carte graphique ne permettait jusqu'à présent que de traiter des sommets et des fragments individuellement, ce qui n'en faisait pas un outil utilisable pour la génération d'images stéréoscopiques. La sortie des *geometry shaders* à la fin de 2007 a changé cela en permettant de manipuler les sommets de chacun des triangles de la géométrie affichée directement sur la carte graphique.

2.3 Notre nouvelle méthode

Actuellement les contenus des environnements virtuels sont de plus en plus riches et détaillés afin d'obtenir un rendu proche du réalisme. La conséquence est que l'affichage se fait à une cadence très proche de la limite du temps réel. Pour une utilisation normale cela ne pose pas de réels problèmes mais lorsqu'il s'agit de faire un rendu stéréoscopique dans un contexte de réalité virtuelle, il ne sera plus possible de maintenir le temps réel.

En générant les images stéréoscopiques sur GPU, il devient possible d'utiliser la stéréoscopie dans la plupart des contextes de réalité virtuelle. Ainsi, l'avantage de favoriser un contenu en relief est que cela va grandement aider l'utilisateur créer le sentiment de présence en stimulant l'immersion.

Pour maintenir le temps réel, notre méthode tire partie d'une caractéristique du rendu stéréoscopique qui est que le contenu de la scène d'une vue à l'autre est identique et qu'il n'y a que le point de vue qui change légèrement. Cette constatation implique que chaque sommet a en commun entre la vue droite et gauche des caractéristiques comme sa position absolue, sa couleur, son illumination diffuse (avec le modèle Lambertien par exemple) ou ambiante... Concrètement chaque sommet conserve ses caractéristiques à partir du moment où celles-ci ne font pas intervenir le point de vue comme l'illumination spéculaire ou les réflexions/réfractions par exemple. Notre algorithme va utiliser ce point clé en faisant usage des *geometry shaders* dont l'emploi se prête justement à ce genre de situation où il est nécessaire de manipuler les sommets de chaque triangle. L'implantation se déroule en deux parties dont l'une se charge de faire la duplication de la géométrie en fonction du point de vue et l'autre de faire le rendu.

2.3.1 Duplication sur GPU

Le principal objectif de l'algorithme est de conserver les opérations traditionnelles qui sont effectuées lors d'un rendu traditionnel, jusqu'au passage dans le *geometry shader*. Cela signifie que dans le programme principal *OpenGL*, il ne doit y avoir aucune modification lors de l'«affichage» de la géométrie : elle ne doit être dessinée qu'une seule fois. Par contre, des transformations liées à l'affichage peuvent être appliquées avant ou après cet appel, comme notamment l'utilisation du programme sur les *shaders* qui va s'occuper de faire le rendu stéréoscopique. Ne pas modifier la géométrie pour la destiner à un point de vue donné implique également que le *vertex shader* ne doit effectuer que des opérations standards comme la récupération des coordonnées de texture ou les normales, qui restent inchangées d'une vue à l'autre. Enfin, la duplication de la géométrie ne peut se faire que sur la position absolue des sommets, or traditionnellement, un *vertex shader* s'occupe de transformer les coordonnées des sommets pour les avoir dans le repère de l'œil. Il faut donc veiller à modifier cette partie.

Le code source 2.1 présente un exemple de *vertex shader* légèrement modifié afin de pouvoir s'adapter à notre algorithme. On notera plus particulièrement à la ligne 7, l'affectation de la position absolue du sommet à la variable interne *gl_Position*. Les autres opérations restent inchangées et ne sont appliquées qu'une seule fois par sommet. Lors d'un rendu stéréoscopique standard ce morceau de code aurait été appelé deux fois pour un même sommet.

Listing 2.1: Code source du *vertex shader*

```
1  varying out vec3 normal;
2
3  void main(void)
4  {
5    normal = normalize(gl_NormalMatrix * gl_Normal);
6    gl_TexCoord[0] = gl_MultiTexCoord0;
7    gl_Position = gl_Vertex;
8  }
```

L'étape suivante peut être décrite comme le cœur de notre algorithme, car c'est elle qui fait pleinement usage du *geometry shader* pour créer le contenu de la seconde vue. Son objectif est, pour chaque triangle appartenant à la géométrie que l'on désire afficher en stéréovision, de récupérer les coordonnées des trois sommets qui le composent et de les copier dans de nouveaux sommets pour finalement leur appliquer les transformations qui correspondent à la vue à laquelle ils sont destinés.

Listing 2.2: Code source du *geometry shader*

```
1  #version 120
2  #extension GL_EXT_geometry_shader4 : enable
3  #extension GL_EXT_gpu_shader4 : enable
4
5  flat varying float out view;
6  uniform mat4 modelviewprojectionmatrix;
7
8  void main(void)
9  {
10   view = 0.0;
11   for(int i =0; i < 3; ++i){
12     gl_Position = gl_ModelViewProjectionMatrix * gl_PositionIn[i];
13     gl_TexCoord[0] = gl_TexCoordIn[i][0];
14     EmitVertex();
15   }
16   EndPrimitive();
17
18   view = 1.0;
19   for(int i =0; i < 3; ++i){
```

```
20        gl_Position = modelviewprojectionmatrix * gl_PositionIn[i];
21        gl_TexCoord[0] = gl_TexCoordIn[i][0];
22        EmitVertex();
23      }
24      EndPrimitive();
25  }
```

On va considérer que la géométrie reçue par le *geometry shader* est celle destinée à la vue de gauche, ainsi dans le listing 2.2, la première itération multiplie les coordonnées des sommets du triangle par les matrices *modelview*[1] et de projection[2] du programme principal *OpenGL*. À chaque sommet traité, un nouveau sommet est émit et finalement le triangle envoyé à l'étape suivante. Pour générer le triangle de la vue de droite, le même principe va être utilisé mais avec cette fois l'utilisation d'une matrice créée spécialement à cette fin. Dans le cas d'un affichage stéréoscopique, la matrice de projection reste généralement inchangée car les paramètres intrinsèques de la caméra (focale, ouverture, ...) sont fixés en début de programme. Par contre, la matrice *modelview* va être modifiée en fonction des changements qui vont être appliqués au point de vue (position, orientation et direction). Cela va également influencer le vecteur qui permet de séparer l'œil gauche de l'œil droit, de la distance inter-oculaire et dont la translation sera introduite dans la matrice *modelview*. La matrice qui va être passée au *geometry shader* afin de calculer les transformations pour la vue de gauche peut alors se résumer à :

Algorithm 2.3.1: CALCUL DE LA MATRICE DE PROJECTION(M)

comment: La matrice de projection M_p est précalculée

for each affichage

do
- Calcul du point de vue gauche
- Déduire le point de vue droit
- Récupérer la matrice *modelview* M_m
- $M \leftarrow M_m \times M_p$
- Transmettre M au *geometry shaders*

Cependant, une fois que les triangles émis ont été rastérisés, l'ensemble des fragments qui vont être reçus dans le *pixel shader* ne vont pas pouvoir être différentiables, c'est-à-dire qu'on ne pourra pas savoir s'il s'agit d'un fragment destiné à la vue de droite ou à la vue de gauche. Pour éviter cela, nous associons une variable à chacun des triangles qui va valoir 0 pour la vue de gauche, et 1 pour la vue de droite. Cette variable est récupérable pour chacun des fragments ce qui permet ainsi de les distinguer.

[1] Matrice qui permet de passer les coordonnées d'un sommet du repère objet à celui de l'œil.

[2] Matrice passant d'un repère 3D à un repère 2D. La projection se fait sur un plan à une distance connue du centre de projection.

La dernière étape constitue donc le traitement de l'ensemble des fragments qui sont générés depuis l'étape précédente. Notre algorithme permet de continuer à appliquer les opérations traditionnelles que l'on retrouve dans ce *shader* comme le calcul de l'illumination ou le texturage. Le but de la méthode est simplement d'envoyer le fragment dans le *buffer* de rendu dédié soit à la vue de gauche, soit à la vue de droite en fonction de la variable mentionnée dans le paragraphe précèdent. Idéalement on obtiendrait alors le code suivant :

Listing 2.3: Code source du *fragment shader*

```
1  #version 120
2  #extension GL_ARB_draw_buffers : enable
3
4  flat varying float in view;
5
6  void main(void)
7  {
8     if(view == 0.0){
9        gl_FragData[0] = gl_Color;
10    }else{
11       gl_FragData[1] = gl_Color;
12    }
13 }
```

Mais nous allons voir dans la prochaine partie que la mise en œuvre de cette méthode implique l'usage d'extensions particulières dont nous ferons mention.

2.3.2 Rendu

L'ensemble des procédés de restitution stéréoscopiques étant variés, mais fondés sur l'exploitation de deux images légèrement différentes, il faut que notre méthode puisse générer le résultat dans deux textures qui pourront être utilisées quel que soit le matériel employé. Le principe est alors d'effectuer la phase de rendu du *geometry shader* dans deux *buffers* séparés qui, dans le programme principal, sont rattachés à deux *framebuffers*[3] distincts et qui sont représentés par des textures. Cela n'est réalisable qu'en employant deux extensions OpenGL spécifiques qui sont les *Framebuffer Objects* aussi connus sous le nom de *FBO*) et *Multiple Draw Target* (aussi connus sous le nom de *MRT*). Les *FBO* sont une extension d'*OpenGL* [Khr08b] qui permet de gérer des contextes supplémentaires à celui initialement présent. L'autre particularité des FBOs est d'offrir la possibilité d'effectuer le rendu dans une texture plutôt que dans un *buffer* propre à *OpenGL*. On retrouve une utilisation courante lors de la récupération d'une carte de profondeur directement dans une texture afin qu'elle puisse être utilisée dans une technique de *shadow mapping* par exemple ce qui accélère grandement les traitements par rapport à une lecture depuis un

[3]Ensemble de *buffers* associés à un contexte donné permettant de stocker le résultat chromatique d'un rendu, une carte de profondeur,...

buffer OpenGL. La seconde extension [Khr08a] *MRT* a été créée afin de pouvoir effectuer le rendu dans plus d'un *buffer* chromatique.

Ainsi pour notre méthode, nous allons créer un nouveau *framebuffer* via un *FBO* dans lequel deux textures dédiées au rendu chromatique seront ajoutées.

L'utilisation de ces extensions impose toutefois un certain nombre de contraintes qu'il faut prendre en compte. En utilisant les *MRT* nous avons l'obligation de rendre les fragments dans chacun des *buffers* car dans le cas contraire la norme décrit le résultat comme indéfini. Une solution est d'utiliser la transparence pour cacher les fragments qui ne devraient pas apparaître en mettant leur valeur *alpha* à 0 et leur composante *RVB* à noir. Ainsi le listing 2.2 devient :

Listing 2.4: Code source du *geometry shader* modifié pour satisfaire la contrainte des MRT

```
1  #version 120
2  #extension GL_ARB_draw_buffers : enable
3
4  flat varying float in view;
5
6  void main(void)
7  {
8     if(view == 0.0){
9        gl_FragData[0] = vec4(gl_Color.rgb,1.0);
10       gl_FragData[1] = vec4(0.0);
11    }else{
12       gl_FragData[0] = vec4(0.0);
13       gl_FragData[1] = vec4(gl_Color.rgb,1.0);
14    }
15 }
```

Pour maintenir la possibilité d'utiliser la transparence de manière normale, le test à appliquer dans le programme principal consiste à éliminer les fragments dont la valeur alpha est égale 0 et uniquement ceux là. Ce processus peut être mis en œuvre en activant l'*alpha test*[4].

Une autre contrainte est liée à l'utilisation couplée des *MRT* et des *FBO* : il n'y a qu'un *buffer* de profondeur commun à chaque buffer de rendu chromatique. Traditionnellement ces extensions sont utilisées ensemble dans le cadre de l'affichage d'une géométrie similaire, mais avec de possibles variations de couleurs ce qui ne pose pas de problème lors du test de profondeur car les informations sont partagées. Dans notre cas, l'emploi d'un *buffer* de profondeur commun va entraîner des conflits et l'apparition d'artefacts lors du rendu car les points de vue ne sont pas les mêmes. Il existe trois solutions qui vont

[4]Ce test permet d'éliminer des fragments en définissant préalablement des conditions sur les valeurs *alpha*

permettre d'outrepasser cette contrainte. La première utilise l'«algorithme du peintre» qui nécessite un ordonnancement de l'ensemble des facettes de la scène en fonction de leur distance par rapport au point de vue. Cette opération peut se révéler assez coûteuse si le nombre de triangles est important. Des problèmes liés à l'algorithme même, peuvent toujours subvenir comme par exemple le cas de deux triangles se chevauchant.

La deuxième solution consiste à faire un rendu dans une seule texture dont les dimensions correspondent à l'union des deux textures de rendu initial. De cette manière, nous nous retrouvons avec un seul *buffer* de rendu chromatique, certes de taille doublée, mais qui offre l'avantage de pouvoir utiliser le *buffer* de profondeur sans contrainte. Cette technique impose cependant quelques manipulations pour que les fragments destinés à la vue de droite soient bien transférés dans la bonne zone et que le *clipping*[5] des deux vues soit correct. Ces modifications ne peuvent avoir lieu que dans le *geometry shader* car le *pixel shader* ne permet pas de modifier la position d'un fragment. Le processus est le même que dans la méthode de départ à la différence qu'une fois que le sommet du triangle a été projeté, on va vérifier qu'il appartient bien à l'espace de projection $(0, 0, w, h)$ où w est la largeur de la fenêtre et h sa hauteur. Si ce n'est pas le cas, on élimine le triangle. Cette solution permet également de se passer de la variable permettant de définir si un triangle appartient à la vue de droite ou à la vue de gauche. Si un triangle appartient à la vue de gauche, nous allons le laisser en l'état, par contre si un triangle appartient à la vue de droite, alors après la projection de chacun de ses sommets, nous leur appliquons une translation horizontale de vecteur $(w, 0, 0)$. Le listing 2.2 est transformé en conséquence :

Listing 2.5: Code source du *geometry shader* modifié pour le test de profondeur

```
1  #version 120
2  #extension GL_EXT_geometry_shader4 : enable
3  #extension GL_EXT_gpu_shader4 : enable
4
5  uniform mat4 modelviewprojectionmatrix;
6  uniform int width;
7  uniform int height;
8
9  void main(void)
10 {
11   vec4 position[3];
12   bool clipping = true;
13   for(int i =0; i < 3; ++i){
14     position[i]=gl_ModelViewProjectionMatrix*gl_PositionIn[i];
15         if(position[i].x>width || position[i].x<0 ||
16        position[i].y<0 || position[i].y>height){
17       clipping == false;
18     }
19   }
20   if(clipping==true){
```

[5]Élimination de la géométrie qui n'est pas présente dans un volume de projection donné

```
21      for(int i =0; i < 3; ++i){
22        gl_Position = position[i];
23        gl_TexCoord[0] = gl_TexCoordIn[i][0];
24        EmitVertex();
25      }
26      EndPrimitive();
27    }
28
29    clipping = true;
30    for(int i =0; i < 3; ++i){
31      position[i] = modelviewprojectionmatrix
32              * gl_PositionIn[i];
33          if(position[i].x>width || position[i].x<0 ||
34         position[i].y<0 || position[i].y>height){
35        clipping == false;
36      }
37    }
38    if(clipping==true){
39      for(int i =0; i < 3; ++i){
40        gl_Position = position[i] + vec4(width,0,0,0);
41        gl_TexCoord[0] = gl_TexCoordIn[i][0];
42        EmitVertex();
43      }
44      EndPrimitive();
45    }
46  }
```

La limite de cette solution est double : d'un côté le fait d'appliquer un *clipping* sur les triangles peut provoquer des disparitions ou des apparitions subites de facettes aux frontières de la zone de rendu (la solution qui consiste à trouver le sous triangle correspondant au découpage serait trop gourmande en temps de calculs), d'un autre côté, il y a la taille de la texture. Selon les caractéristiques des cartes graphiques, la taille maximale des textures utilisables est variable. Il se peut donc que certaines cartes graphiques ne permettent pas de doubler la taille de la texture.

La troisième solution pourrait être qualifiée d'anticipation, car elle n'est pas applicable pour le moment, mais pourrait le devenir dans un avenir proche avec le développement de nouveaux pilotes pour les cartes graphiques. Le principe de cette solution est de faire notre propre test de profondeur dans le *geometry shader* en enregistrant la profondeur d'un fragment dans sa valeur *alpha*. Pour chaque nouveau fragment, il suffit de comparer sa profondeur à celle stockée dans la texture de rendu. En conséquence, il est soit éliminé soit écrit dans la texture de rendu à la place du précèdent. C'est ce mécanisme de lecture et d'écriture dans une texture active qui n'est malheureusement pas encore disponible.

2.4 Résultats

Pour évaluer les performances de notre nouvelle méthode, nous avons effectué une série de tests sur un *PC Core 2 DUO 2,4Ghz* sous *Linux* avec une carte graphique *NVidia 8800GTX* comprenant *768Mb* de mémoire. La scène, sur laquelle nous appliquons la stéréoscopie sur GPU, consiste à afficher plusieurs fois le modèle de lapin créé par l'université de Stanford (environ 70000 triangles).

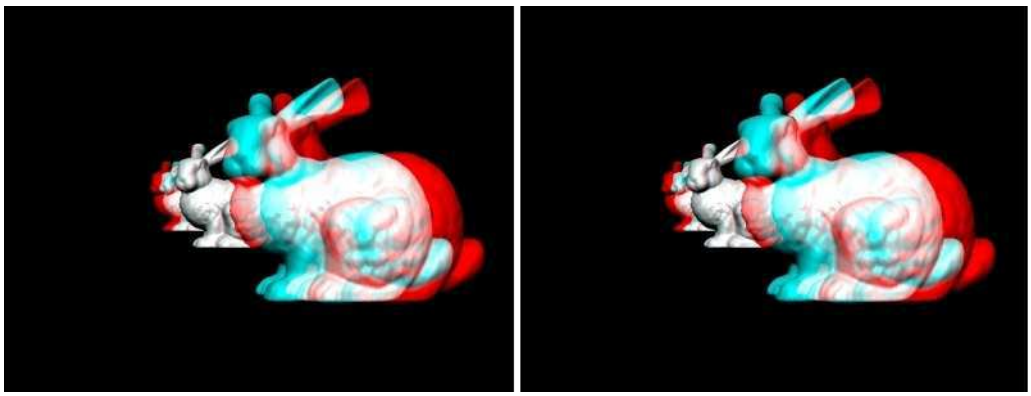


FIG. 25: Résultat obtenu par notre méthode (gauche) comparé à celui de la méthode traditionnelle (droite)

Les «Stanford bunnies» sont positionnés à différentes profondeurs pour bien vérifier le résultat des décalages comme présenté sur la figure 25. La table 2.1 présente les résultats obtenus dans différentes situations d'utilisation.

	Méthode traditionnelle			
Nombre de triangles	70000	210000	350000	630000
Rendu sans lumière (img\s)	163	55	33	18
Illumination par sommet (img\s)	83	28	16	9
Illumination par pixel (img\s)	82	27	16	9
	Notre méthode			
Nombre de triangles	70000	210000	350000	630000
Rendu sans lumière (img\s)	304	108	65	36
Illumination par sommet (img\s)	158	54	33	19
Illumination par pixel (img\s)	156	54	33	19

TAB. 2.1: Comparaison de notre méthode à un rendu traditionnel

D'une manière générale, nous pouvons constater qu'en comparaison à la méthode de rendu stéréoscopique traditionnelle en deux passes, notre optimisation permet d'accélérer les temps de rendu par deux. Notre objectif étant de proposer un regroupement des opérations au niveau du traitement des sommets, on pourrait s'attendre à un gain plus

important pour l'illumination par sommet par rapport à celle sur pixel. Toutefois, les résultats démontrent que nous obtenons à peu près les mêmes résultats. Ce fait est la conséquence de l'apparition d'une architecture unifiée sur les cartes graphiques récentes qui permet de repartir les ressources en allouant plus ou moins d'unités de calculs à un *shaders*.

En ce qui concerne les limites techniques associées à notre méthode comme le *buffer* de profondeur non partagé, nous attendons la sortie de nouvelles extensions qui pourraient remédier à cela, notamment avec la prochaine venue de la version 3.0 d'*OpenGL*.

2.5 Application à l'autostéréoscopie

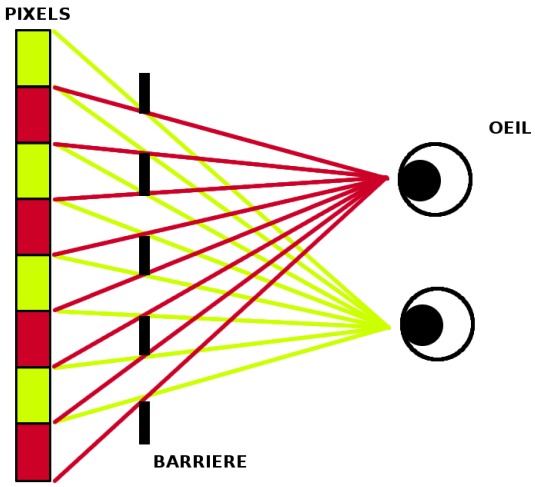
Depuis quelques années le procédé de restitution stéréoscopique, nommé écran autostéréoscopique, se développe de plus en plus. Ce principe a de multiples avantages sur tous les autres. Tout d'abord, la restitution n'est pas mono-utilisateur, car plusieurs utilisateurs peuvent visionner la scène en relief depuis la position où ils se trouvent. Ensuite depuis chaque point de vue, l'observateur a une vision de la scène virtuelle correspondant à sa position sans que cela ne nécessite de *tracking*. Enfin ce procédé a l'énorme avantage de proposer une solution qui ne nécessite aucune interface de visualisation supplémentaire comme des lunettes. Ce point, comparé aux solutions stéréoscopiques existantes, offre un réel bénéfice en réalité virtuelle pour maintenir le sentiment de présence chez l'utilisateur car ce dernier est soulagé des classiques interfaces, que nous pouvons qualifier d'envahissantes, qui peuvent nuire à l'immersion.

La génération actuelle d'écrans permet d'obtenir jusqu'à neuf points de vue différents, offrant à l'observateur une liberté suffisante pour apprécier l'effet. Cependant, la génération de neuf vues simultanées nécessite beaucoup plus de calculs que pour un rendu stéréoscopique traditionnel, d'où l'intérêt d'utiliser notre méthode. Dans cette section nous allons commencer par présenter les technologies associées à l'autostéréoscopie ainsi que les méthodes existantes liées au rendu. Nous décrirons ensuite les modifications apportées à notre méthode pour satisfaire les nouvelles conditions avant d'en présenter les résultats.

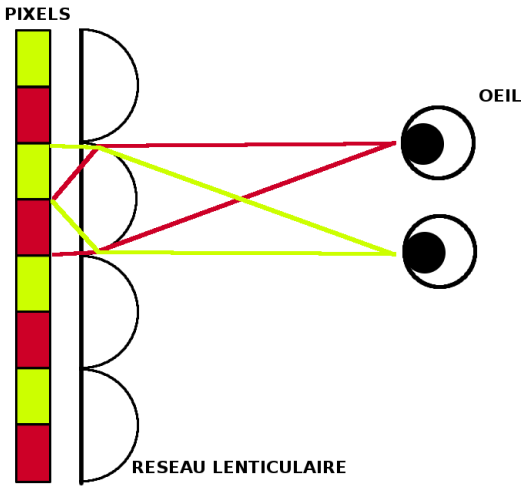
2.5.1 Etat de l'art

Le procédé utilisé pour les écrans autostéréoscopiques [Rob03, Oko80] existe depuis le début du vingtième siècle, mais c'est grâce à l'évolution en matière d'écrans *LCD* qui a eu lieu ces dernières années que cette technique de restitution a pu entrer dans une phase de production industrielle. Deux technologies, [SHA06] et [Ele06], se partagent le marché. On trouve d'un côté les écrans basés sur de fines lamelles qui permettent, depuis un point de vue précis, de ne voir que certaines lignes verticales de pixels. Le principe de fonctionnement de ces écrans, dits à *barrière parallaxe* et dont les ancêtres sont [Ive03] et [Ive36], est schématisé dans la figure 26a. Cet écran stéréoscopique a malgré tout quelques lim-

ites [Wid01] car l'utilisation de lamelles pour occulter certains pixels entraîne une baisse de la luminosité de l'affichage et il semblerait que le faible écart entre les lamelles produise des phénomènes de diffraction. En 2005 il était toutefois possible de trouver de tels écrans de 127cm dans les hypermarchés *Carrefour* qui diffusaient des spots publicitaires. Toutefois, ils engendraient rapidement, chez certaines personnes, un mal de tête.



(a)



(b)

FIG. 26: Illustration du fonction d'un écran à barrière parallaxe *a* et d'un écran avec un réseau lenticulaire *b*

L'autre procédé de restitution est appelé *réseau lenticulaire*. Comme son nom l'indique, il se fonde sur une surface, composée de minuscules bandes de lentilles, qui vient se «coller» sur la dalle de l'écran. Chacune de ces lentilles va recouvrir un nombre bien défini de pixels et l'observateur, en fonction de sa position, ne recevra l'image provenant que d'une seule série de pixels. La composition des pixels suit un schéma bien précis avec par exemple un agencement de lignes parallèles (figure 27a) ou un agencement oblique (figure 27c) qui

permet d'augmenter la résolution horizontale du rendu au dépend de la résolution verticale. Les écrans actuellement disponibles offrent la possibilité de restituer jusqu'à neuf vues mais des recherches sur une déclinaison de ces écrans permettent d'obtenir jusqu'à 64 vues [Tak06]. Le principe général de fonctionnement de type d'écran autostéréoscopique est donné dans la figure 26b.

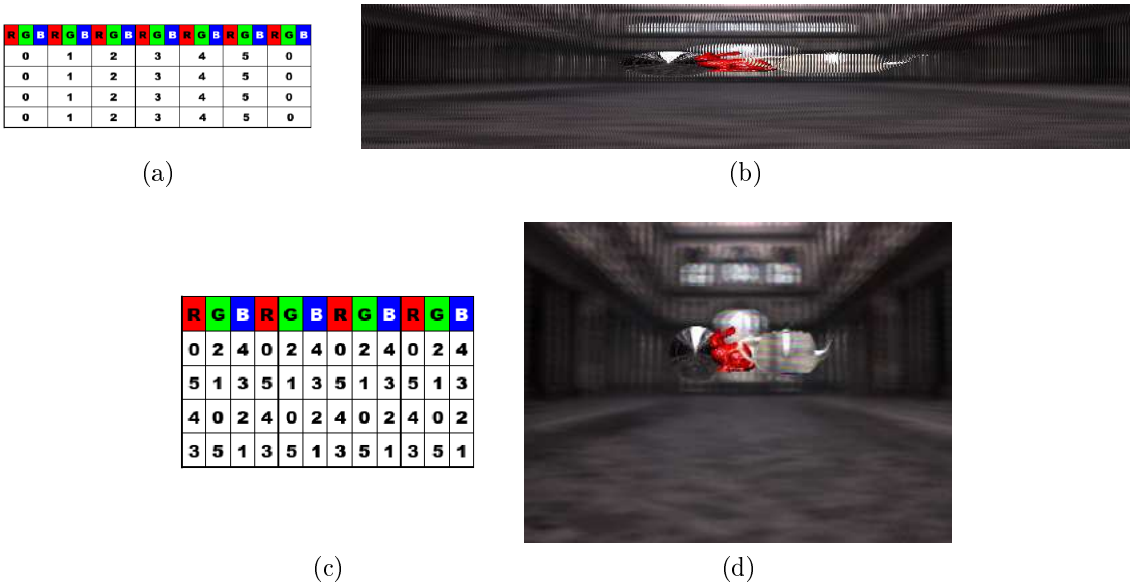


FIG. 27: Le tableau *a* présente un agencement vertical des vues avec un exemple de rendu sur l'image *b*. Le tableau *c* présente un agencement des vues en oblique avec son illustration dans l'image *d*

La génération des différentes vues nécessaires à un rendu sur écran autostéréoscopique nécessite une bande passante suffisante entre les unités de calcul et d'affichage pour que la transmission des données puisse se faire le plus rapidement possible afin d'obtenir un minimum de 25 images par seconde. C'est dans l'optique de rendre cette partie plus performante que Kaufmann et Akil [KA06] ont proposé une architecture dédiée associée à un algorithme se basant sur la «voxelisation» de la scène qui permet d'encoder et décoder 200 images à une cadence de 30 images par seconde. La même limite intervient dans le cadre de l'affichage sur un écran autostéréoscopique à partir de plusieurs sources vidéo (on parle aussi de 3D TV). Mais cette fois c'est l'acquisition vidéo qui pose problème, car une unité de calcul peut saturer si les flux à gérer sont trop nombreux. Nozick et Saito [NS07] proposent une solution se fondant sur un nouvel algorithme de rendu à base de vidéo en direct (*Online Video-Based Rendering*) ne nécessitant que quatre sources vidéo et permettant de générer automatiquement les vues manquantes.

Une autre contrainte est de pouvoir rendre le maximum de points de vue en conservant un affichage en temps réel et c'est à cette fin que notre algorithme se destine. Parmi les méthodes existantes, une première est proposée par Hübner *et al* [HZP06] pour effectuer la génération d'images pour plusieurs points de vue en une seule passe dans un contexte

de rendu à base de points (*Point-Based Rendering*). Les deux solutions qu'ils proposent font pleinement usage des capacités des *GPU* pour calculer les intersections entre le rayon émis depuis le point de vue et les *splats*[6], soit en combinant le *vertex shader* (calcul des coordonnées paramétriques pour les différentes vues) avec le *fragment shader*, soit uniquement dans le *fragment shader*. Cette dernière se révèle la plus efficace avec des résultats pour huit vues qui n'ont besoin que de trois fois plus de temps de calcul au lieu de huit fois plus. En 2007, Hübner *et al* proposent une adaptation de leur précédente méthode en l'appliquant au rendu volumique direct afin de produire plusieurs vues en une seule passe [HZP07]. Elle se fonde sur une corrélation entre les différentes images à partir des informations issues d'une seule projection en appliquant les calculs sur GPU. Cette optimisation permet de réduire par deux les temps de traitement par rapport à un rendu multi-vues traditionnel.

(a) (b) (c)

FIG. 28: Illustration du format 2D+Z : *(a)* l'image de référence ; *(b)* la carte de profondeur associée ; *(b)* Le résultat de la méthode *Copyright : Christoph Fehn*

Une autre méthode proposée par Fehn [Feh03] en 2003 consiste à associer une image 2D et une carte de profondeur pour reconstituer les autres images des points de vue situés à sa gauche et à sa droite. Ce procédé, appelé «format 2D+Z » ou 2.5D va, pour chaque pixel du point de vue que l'on souhaite générer, calculer l'information chromatique à récupérer dans la vue de référence en fonction de la carte de profondeur. Cette technique atteint toutefois ses limites lorsque des occlusions présentes dans l'image de référence n'existent pas dans l'image générée, car nous n'avons pas les données nécessaires pour remplir les trous. Une solution est d'appliquer un lissage via un filtre gaussien 2D simple qui dépend de la profondeur afin de remplir ces zones où l'information est manquante. Les autres solutions évoquées font mention d'interpolations sur les composantes chromatiques ou alors d'une extrapolation de l'information depuis l'arrière plan de l'image.

Une solution autostéréoscopique de la firme *Philips* [Phi06] emploie un format *WOWvx Declipse* qui est un dérivé du précèdent. Cette fois, en plus de l'image et de sa carte de profondeur, on envoie une image de l'arrière plan avec également sa carte de profondeur, comme illustré dans la figure 29. De cette manière, il est possible de remplir les zones où l'information était manquante. Par contre, cela nécessite que l'arrière plan soit statique et

[6]Sous-élement 2D généralement ellipsoïdale utilisé pour représenter chacun des points d'une surface

que les scènes soient relativement épurées afin d'éviter au maximum les occlusions inter-objets.

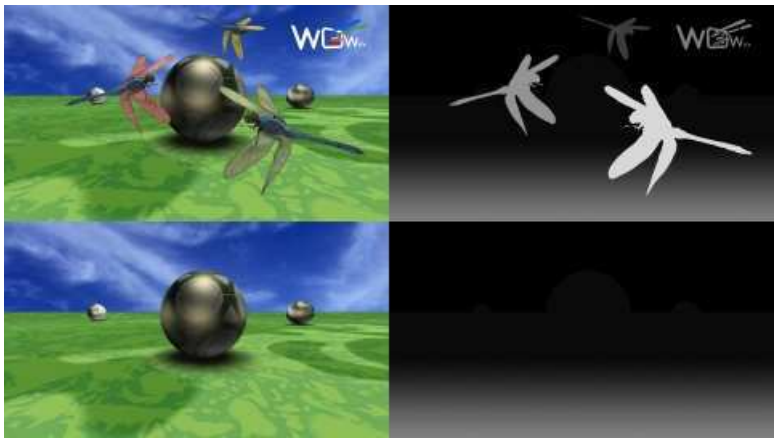


FIG. 29: Les images utilisées dans le format *WOWvx Declipse Copyright : Philips Electronics*

Parmi cet ensemble de méthodes que nous venons de présenter, aucune ne permet de vraiment résoudre notre problème de rendu austéréoscopique en temps réel. La méthode de Akil et *al.* [KA06] est trop spécifique, car elle nécessite d'une part de stocker les données dans une structure bien particulière et d'autre part nécessite l'utilisation d'une architecture matérielle dédiée. Finalement, elle peut difficilement s'appliquer au appareils autostéréoscopiques actuellement en vente. La solution de Nozick et Saito [NS07] repose, quant à elle, seulement sur des sources vidéos et s'adapte difficilement à notre cas. Hübner *et al* [HZP06, HZP07] ont proposé un ensemble de méthodes permettant de générer plusieurs points de vue en une seule passe en temps réel. Toutefois, les conditions de rendu à base de points ou volumiques ne correspondent pas à notre volonté de conserver la simplicité du programme initiale, c'est-à-dire, demandant très peu de modifications par rapport à un rendu monoscopique simple. De plus, ce type de méthodes produit des artefacts lors de l'affichage, causés par l'utilisation d'un lissage ou d'une extrapolation de l'information comme dans la solution proposée par Fehn [Feh03] et la firme *Philips* [Phi06].

2.5.2 Algorithme de rendu

Notre méthode, décrite dans la section 2.3.1, peut aisément être transposée d'un contexte stéréoscopique à un affichage sur écran autostéréoscopique, ceux-ci pouvant actuellement afficher jusqu'à neuf vues, l'algorithme (figure 30) peut alors être généralisé à :

Algorithm 2.5.1: AUTOSTÉRÉOSCOPIE(M, N)

comment: N est le nombre de points de vue

comment: Les matrices de projections $M_p(i)$ avec $0 \leq i \leq N$ sont précalculées

for each affichage

do for $i \leftarrow 0$ **to** N

do
- Cloner la primitive
- Calculer la matrice $modelview M_m(i)$
- $M \leftarrow M_m(i) \times M_p(i)$
- **output** $(M \times primitive)$

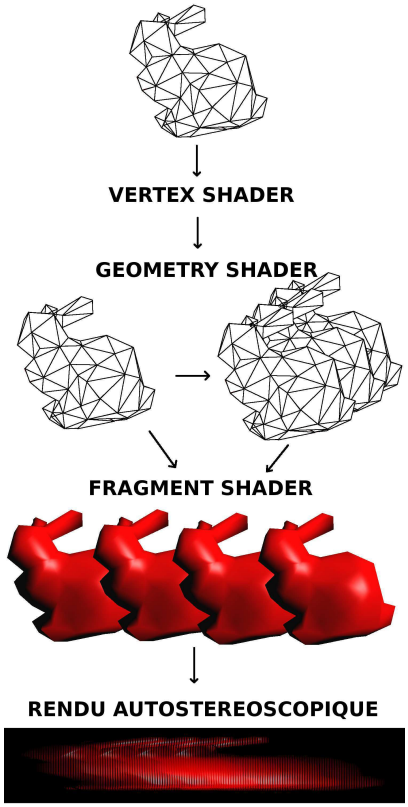


FIG. 30: Schématisation de l'algorithme de rendu multi-vues par *GPU*

Le code *GLSL*, associé à cette nouvelle version de la méthode, est présenté dans le listing 2.6 pour le *geometry shader* et le listing 2.7 pour le *fragment shader*. Les principales différences résident dans l'ajout d'une variable *nview* qui permet de récupérer le nombre de vues à rendre, mais dont ce dernier est limité par la constante *MAX_ VIEW*. L'émission des nouveaux triangles est fondée sur une boucle conditionnelle comme présentée dans le code de la méthode. Toutefois, l'algorithme est présenté de cette manière afin d'en clarifier sa compréhension car les *shaders* sont réputés comme connaissant des baisses de performance lorsque qu'il y a ce genre d'instructions à gérer et se révèlent plus efficace si

les boucles sont déroulées à la main.

Listing 2.6: Code source du *geometry shader* modifié pour le test de profondeur

```
#version 120
#extension GL_EXT_geometry_shader4 : enable
#extension GL_EXT_gpu_shader4 : enable

const int MAX_VIEW = 8;
flat varying out float view;
uniform int nview;
uniform mat4 matrix[MAX_VIEW];

void main(void)
{
 for(int v=0; v<nview; ++v){
   flag = float(v);
   for(int i=0; i<3; ++i){
    gl_Position = matrix[v] * gl_PositionIn[i];
    EmitVertex();
   }
   EndPrimitive();
 }
}
```

Listing 2.7: Code source du *geometry shader* modifié pour le test de profondeur

```
#version 120
#extension GL_ARB_draw_buffers : enable

flat varying in float view;
uniform int nview;

void main()
{
  for(int i=0;i<nview;++i){
   if(float(i)==view)
    gl_FragData[i]=vec4(colorfrag.rgb,1.0);
   else
    gl_FragData[i]=vec4(0.0);
  }
}
```

Outre les problèmes rencontrés avec le test de profondeur qui restent toujours présents, l'utilisation d'un grand nombre de *buffers* de rendu peut poser problème. Actuellement le nombre de textures de rendu autorisées dans un *FrameBuffer Object* est matériellement

limité à huit (du moins sur les cartes graphiques avec *OpenGL*). C'est pour cette raison que le nombre de vues dans le listing 2.6 est limité par une constante. Sur une grande variété d'écrans auto-stéréoscopiques le nombre d'images requises est supérieur à ce maximum, il est donc nécessaire de trouver une alternative. Mis à part l'attente d'une évolution des cartes graphiques permettant d'éliminer cette contrainte, une solution temporaire consiste à ne plus restreindre le nombre de passes à une dans le programme principal, afin d'augmenter le nombre de textures de rendu. Par exemple, pour pouvoir obtenir neuf vues, il est possible d'utiliser deux *FBO* et dans une première passe rendre huit vues et dans une seconde, ne rendre que celle qui manque. Cependant, nous verrons dans la section suivante que cette solution permet de conserver de bonnes performances comparativement à la méthode traditionnelle de rendu.

2.5.3 Résultats

Les résultats de notre méthode ont été obtenus en employant une scène composée d'environ 80000 triangles et faisant globalement intervenir différents effets visuels tels que l'utilisation de textures d'environnement afin d'obtenir des réflexions ou des réfractions, ainsi qu'une illumination par sommet avec réflexions spéculaires. L'utilisation de ces derniers permet de prouver que notre méthode est opérationnelle quel que soit le rendu souhaité. La machine, qui a servi aux tests, est un *PC Core 2 DUO 2,4Ghz* sous *Linux* avec une carte graphique *NVidia 8800GTX*. Un exemple visuel de ce résultat est illustré dans la figure 31.

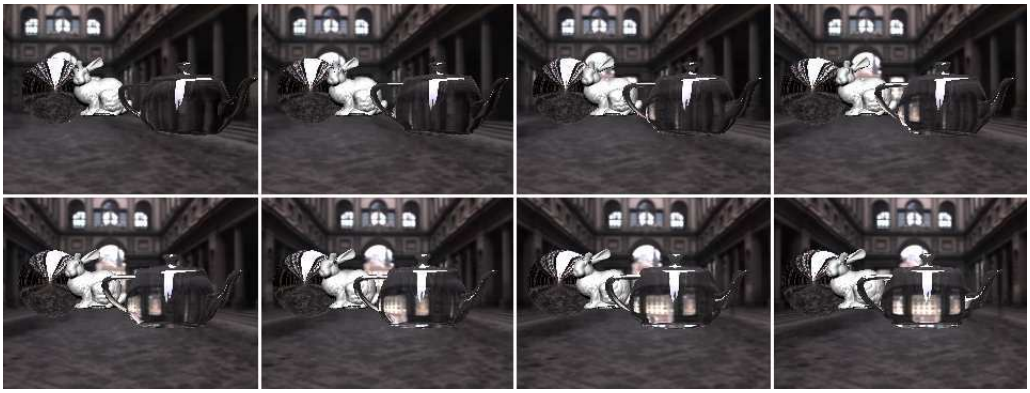


FIG. 31: Résultats obtenus lors de la génération de huit vues avec notre méthode

Dans un premier temps, l'évaluation des performances en terme de rapidité de rendu s'est focalisée sur une génération d'un maximum de huit points de vue et a été comparée à un rendu multi-vues traditionnel. La table 2.2 montre que les résultats obtenus en terme de nombre d'images par seconde restent supérieurs à ceux d'un rendu «normal». Cela s'explique par l'économie de calculs fait au niveau des sommets qui permet d'éviter des redondances lors de leur traitement avec par exemple le calcul des coordonnées de texture ou de leur couleur. De plus, nous pouvons remarquer que faire le rendu en une seule passe

sur la carte graphique permet de diminuer l'échange d'informations entre le CPU et le GPU, d'où une amélioration des performances.

Nous noterons plus particulièrement que la méthode est efficace avec un rendu pour deux et quatre points de vue où l'on obtient un temps de rendu pratiquement divisé par deux. Toutefois, plus le nombre d'image à générer est élevé plus les performances de notre méthode tendent vers celles de la méthode traditionnelle. Cette tendance est dûe au fait que les gains obtenus par la factorisation des calculs lors du *vertex shader* sont trop faibles pour compenser les calculs effectués dans le *pixel shader*. L'autre cause possible peut venir du *geometry shader* qui, lorsqu'il y a un grand nombre de vues à générer, doit émettre une quantité importante de triangles. Cette première implémentation des *geometry shaders* n'étant, d'après la communauté graphique, pas encore très efficace on pourrait donc s'attendre à des améliorations dans les prochaines années.

Durant nos évaluations, nous avons pu constater que le simple fait d'ajouter une variable dans le *geometry shader*, pour transmettre une donnée au fragment shader, pouvait entraîner un doublement des temps de calcul. Cela vient donc confirmer que pour le moment cette implémentation des *geometry shaders* n'est pas optimale.

Nombre de vues	1	2	4	8
Rendu traditionnel (img\s)	157	80	38	19
Notre méthode (img\s)	157	141	70	24

TAB. 2.2: Comparaison de notre méthode à un rendu traditionnel

La deuxième série de tests a pour objectif de confronter notre méthode à diverses situations d'utilisation faisant intervenir différents types de *shaders*. Les résultats sont présentés dans la table 2.3. Nous pouvons constater que les performances restent cohérentes sur le principe de répartition de la charge de calcul entre le vertex shader et le fragment shader. Ainsi il n'y a quasiment pas de différence entre une illumination par sommet ou par pixel.

Nombre de vues	2	4	6	8
Sans illumination (img\s)	326	168	80	60
Textures (img\s)	326	167	80	60
Illumination par sommet (img\s)	160	83	28	23
Illumination par pixel (img\s)	160	83	28	24

TAB. 2.3: Performances obtenues en fonction de différents types de colorisation des méthodes

Enfin nous avons testé les performances de notre méthode de génération multi-vues par *GPU* en faisant varier le nombre de passes et le nombre de vues générées par passe.

La table 2.4 présente ces résultats avec la volonté de générer douze vues, ce qui est actuellement impossible avec les cartes graphiques courantes. Nous pouvons voir qu'il existe effectivement une corrélation entre le nombre de passes et le nombre de vues. On peut dès lors en conclure que le meilleur rendement est obtenu avec trois rendus par passe. Avec trois vues on atteint donc l'équilibre entre le gain obtenu au niveau du *vertex shader* et les limites liées à la faiblesse du *geometry shader* lors d'une montée en charge du nombre de primitives à traiter. Par exemple, dans le cas de la génération de huit vues réparties sur quatre passes de deux vues chacune, nous pouvons constater une amélioration des performances de 24 images par secondes à 28 images par seconde (soit quasiment 10 images par seconde de mieux que la méthode traditionnelle).

Nombre de passes	1	2	3	4	6
Nombre de vues par passe	12	6	4	3	2
Vitesse de rendu (img\s)	N/A	14	15	23	21

TAB. 2.4: Génération de douze rendus depuis douze points de vue en modifiant le nombre de passes et le nombre de vues par passe

3

Aide à la visualisation stéréoscopique

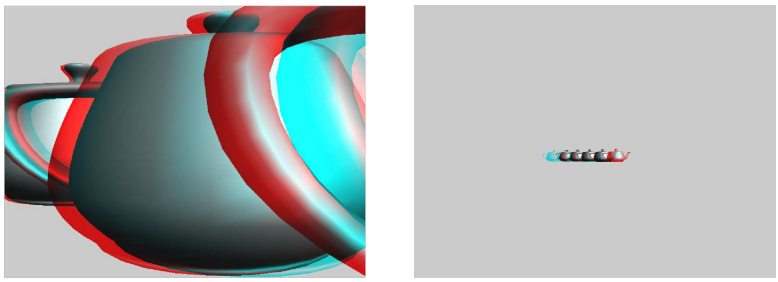
3.1 Objectifs

Lorsqu'un observateur utilise un système de réalité virtuelle, il doit faire face à deux types de contraintes liées au contexte visuel. La première est induite par le contenu de la scène virtuelle qui peut être plus ou moins riche en éléments. Lorsque l'utilisateur essaye de percevoir un environnement de ce type, il va avoir tendance à ne pas trouver d'objet sur lequel porter son attention et va par conséquent laisser vagabonder son regard dans la scène. On peut faire une analogie avec l'exploration d'une photographie qui s'effectue culturellement de la gauche vers la droite et du haut vers le bas comme le décrit la figure 32. Il est à noter que certains éléments peuvent naturellement prendre le devant sur d'autre, ainsi le regard a tendance à se focaliser sur les zones complexes d'une image, les formes grandes ou proches et plus particulièrement sur le centre de l'image [Pra06]. De plus parmi les autres éléments qui attirent le regard nous pouvons citer la netteté, la régularité, le premier plan, les couleurs chaudes... Enfin les personnages sont aussi de bons attracteurs du regard et plus spécifiquement les visages sur lequel un observateur aura tendance à se focaliser en premier.

FIG. 32: Parcours traditionnel effectué par le regard pour percevoir le contenu d'une photographie

La deuxième contrainte concerne plus spécifiquement les images stéréoscopiques. Leur

observation peut provoquer, chez l'utilisateur, l'apparition d'une fatigue oculaire pouvant même aller jusqu'à causer des maux de tête. Cet effet de bord est principalement provoqué par le phénomène d'accommodation-convergence. Concrètement, lorsque nous regardons une image stéréoscopique, nos yeux vont avoir tendance à faire la mise au point sur la surface où se situe l'image et en même temps converger vers un point de l'espace différent. Cette ambiguïté optique, à l'origine des fatigues oculaires particulièrement chez les utilisateurs occasionnels, peut alors être néfaste à la création du sentiment de présence lors d'une expérience de réalité virtuelle. Dans la pratique, ce phénomène se produit dans les zones de l'image où la disparité d'un objet est trop important (cas d'un relief ou d'une profondeur très prononcée comme sur la figure 33a).



(a) (b)

FIG. 33: Différence d'écart sur une image stéréoscopique (anaglyphe) dans le cas *(a)* d'un relief ou *(b)* d'une profondeur

Une solution peut passer par l'emploi d'une méthode de flou avec l'objectif d'occulter des éléments particuliers tout en laissant les autres nets. De cette manière il devient possible de mettre en avant certains objets de la scène en rendant les autres difficilement identifiables. Ce choix implique que l'observateur aura une plus forte tendance à se concentrer sur les parties nettes et ainsi évitera, par exemple, de poser son regard sur des objets de second plan dont il aurait été difficile de reconstruire l'effet de relief à cause du trop grand écart entre les deux vues. La figure 34 illustre la différence qu'il y a entre l'utilisation ou non, d'un flou sur une photo pour mettre en avant un élément bien précis.

(a) (b)

FIG. 34: Exemple sans *(a)* et avec *(b)* l'utilisation du flou dans un contexte photographique pour mettre en avant certains élèments de l'environnement

Il existe deux approches permettant d'obtenir ce résultat. La première méthode, appelée *blur*, va appliquer un «brouillage» sur chaque élément du décor devant être flouté. Cela a pour effet d'atténuer les contours et mélanger les couleurs. La technique utilisée fait généralement appelle à une matrice de convolution[7] dont la taille et les composantes vont influencer la forme (circulaire, carré,...) et l'intensité du flou. La figure 35 présente différents résultats de *blur* selon la matrice de convolution employée.

(a)

(b)

(c)

(d)

(e)

FIG. 35: Différentes applications d'un *blur* sur l'image originale *(a)* : *(b)* flou gaussien de taille 20x20, *(c)* flou gaussien de taille 40x40, *(d)* matrice de convolution «circulaire» 5x5 et *(e)* matrice de convolution carrée 5x5

La seconde approche consiste à utiliser un flou bien connu des amateurs de photographie qui est le flou de profondeur de champ dont un exemple est illustré sur la figure 36.

[7]Matrice qui décrit la relation entre les valeurs des pixels de sortie et celle des pixels d'entrée

Dans un système optique simple, la lentille va avoir pour effet de faire converger les rayons lumineux vers un point de l'axe optique. Lorsqu'il s'agit de rayons parallèles (source lumineuse à l'infini), la convergence se produit en un point particulier appelé point de focale ou foyer. Dans les autres cas, la convergence a lieu sur un point situé derrière le point de focale (figure 37). La relation entre ce point image et son origine est donnée par :

$$\frac{1}{D} + \frac{1}{V} = \frac{1}{F}$$

FIG. 36: Exemple de profondeur de champ en photographie

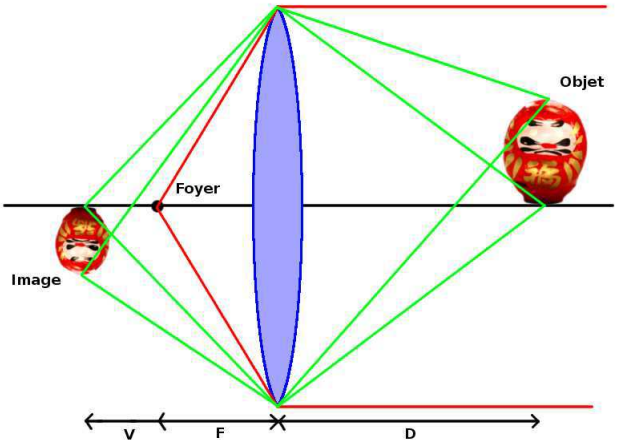


FIG. 37: Illustration du fonctionnement d'une lentille mince

À ce système optique peut venir s'ajouter un plan image sur lequel vient s'imprimer les rayons [PC82]. Si le point image ne se projette pas sur le plan image, il va alors se créer une tâche appelée cercle de confusion qui va être à l'origine du flou (figure 38). En pratique l'œil perçoit les éléments nets jusqu'à ce que le cercle de confusion ait atteint une taille donnée. La profondeur de champ est alors définie comme étant l'espace que nous percevons comme net. Le diamètre du cercle de confusion est influencé par deux

paramètres qui sont la distance focale[8] et l'ouverture[9]. Ainsi la profondeur de champ est petite lorsque la taille minimale du cercle de confusion est petite, lorsque la distance focale est grande et/ou lorsque l'ouverture est grande.

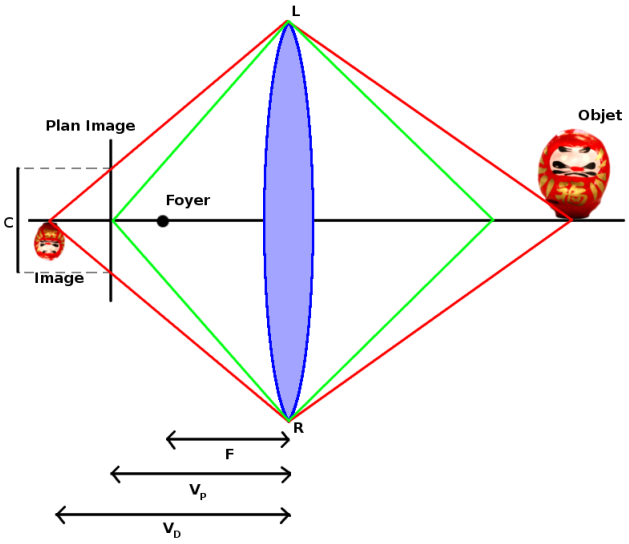


FIG. 38: Illustration du concept de cercle de confusion

Pour reproduire ce genre d'effet optique dans le contexte de la synthèse d'image, Potmesil et Chakravarty [PC82] ont proposé une généralisation du modèle de caméra de type sténopé couramment utilisé en informatique graphique à un modèle prenant en compte la présence d'une lentille mince. Ils décrivent ainsi comment il est possible de calculer la taille du cercle de confusion dans ce genre de système optique. Pour une distance focale et une ouverture données on obtient à l'aide des relations géométriques du schéma de la figure 38 :

$$C = |V_D - V_P| \frac{LR}{V_D}$$

avec C le diamètre du cercle de confusion et $LR = F/n$ où n représente l'ouverture.

En ajoutant du flou de profondeur de champ aux images de synthèse stéréoscopiques, nous allons contribuer à améliorer certains des quatre piliers, utilisés pour créer le sentiment de présence chez l'utilisateur. Tout d'abord, le flou va permettre de limiter les fatigues oculaires, causées par le phénomène d'«accommodation-convergence», ce qui va favoriser l'extension de la durée des expériences de réalité virtuelle et, par conséquent, l'immersion de l'utilisateur. En proposant d'utiliser le flou pour focaliser l'attention de l'utilisateur, nous allons pouvoir stimuler l'intérêt de l'utilisateur vers l'environnement virtuel et ainsi éviter les perturbations provenant de l'extérieur du système de réalité virtuelle (bruits parasites, câblages, présence d'autres personnes,...). Enfin, selon le contexte d'application, le flou va permettre de développer une autre d'interaction entre

[8]Distance entre le centre optique et le foyer

[9]Espace circulaire placé devant la lentille permettant de contrôler la quantité de lumière entrante. On parle aussi de diaphragme

l'univers virtuel et l'utilisateur en proposant un positionnement du plan de netteté en fonction d'une situation précise.

3.2 État de l'art

Les différentes techniques visant à ajouter du flou dans les images de synthèse qui ont été proposées ces dernières années, sont assez variées. Cependant, l'objectif de leur utilisation reste globalement le même : donner plus de crédibilité, voire de réalisme, aux images de synthèse en y ajoutant ce genre d'effet d'optique. C'est pourquoi on retrouve de plus en plus de flou de profondeur de champ dans plusieurs productions cinématographiques de chez *Pixar* ou *Dreamworks* ou alors dans divers jeux vidéo orientés «Tir à la première personne».

Les différentes méthodes de simulation de flou peuvent être cataloguées en deux catégories. On trouve tout d'abord les algorithmes qui produisent un flou qui est visuellement très réaliste, mais qui nécessite généralement des temps de rendu longs soit à cause de la complexité même des calculs dû par exemple à la volonté de reproduire la physique associée à une optique, soit parce que c'est la méthode de rendu employée qui est lente.

Ensuite on trouve les méthodes cherchant à simuler du flou en temps réel, mais dans ce cas c'est la qualité même du rendu qui se voit diminuée. Malgré tout c'est principalement grâce à l'arrivée des *shaders* que ces méthodes ont pu connaître un développement plus important.

La présentation de l'état de l'art va s'organiser en fonction du temps de rendu nécessaire pour obtenir du flou dans les images de synthèse. Ce choix est principalement motivé par le fait qu'en réalité virtuelle, l'affichage doit se faire en temps réel pour que la partie visuelle du système ne soit pas un frein à la création d'un sentiment de présence. Enfin, cette partie se terminera par la présentation de différentes applications du flou dans un contexte de réalité virtuelle.

3.2.1 Simulation de flou non temps réel

Accumulation

Une manière simple de générer du flou en synthèse d'images est d'utiliser plusieurs images [HA90, ND93] ayant des points de vue légèrement différents d'une même scène. En mélangeant toutes ces images, il va alors être possible de produire un flou (figure 39). Ce dernier est parfaitement paramétrable selon la manière dont les «caméras» sont positionnées. Tout d'abord, le point de visée a une très grande importance, car il va permettre de définir le point de netteté dans l'image finale. Trois cas sont possibles :

- Les caméras pointent dans la même direction : apparition d'un «blur» sur l'ensemble de l'image

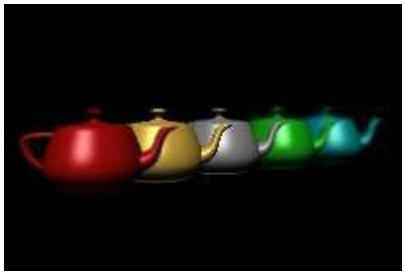


FIG. 39: Flou obtenu en utilisant le buffer d'accumulation *Copyright : Jackie Neider et Tom Davis*

- Les caméras pointent dans des directions diffèrentes : pas de résultat
- Les caméras pointent vers un même point de l'espace : zone de netteté autour de ce point (figure 40)

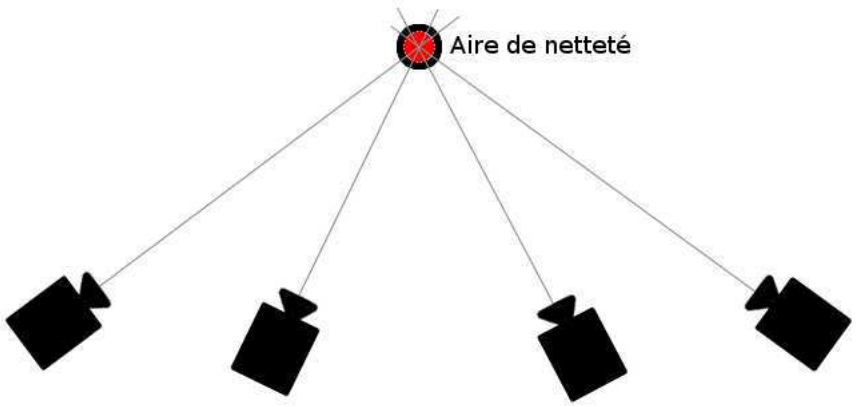


FIG. 40: Le flou d'accumulation est obtenu en faisant coverger les axes optiques des caméras vers un même point

Il y a également une importance sur les décalages entre les différents points de vue. Si les différentes caméras sont séparées par une très faible distance, alors le flou aura tendance à être très faible, rendant ainsi la zone de netteté assez étendue. Dans le cas contraire le flou sera rapidement très prononcé induisant une zone de netteté réduite. Enfin le nombre d'images en entrée de la méthode va directement influencer la qualité du rendu mais également les performances en raison du nombre de passes de rendu nécessaires. Cependant, la définition de l'ensemble des paramètres de cette méthode d'accumulation reste empirique.

Notons qu'une version étendue de cette méthode a été proposée par Heidrich *et al* [HSS97] dans le but d'accroître la véracité de la physique du flou en calculant précisément la position que doivent avoir les points de vue. Pichard *et al* [PMT05] ont, quant à eux, proposé une méthode employant le *buffer* d'accumulation pour améliorer

la qualité visuel du flou en tenant compte de la forme du diaphragme et des écarts de luminosité, comme illustré sur la figure 41.

FIG. 41: Simulation de la profondeur de champ tenant compte de la forme du diaphragme *Copyright : Cyril Pichard*

Malgré la présence d'un *buffer* d'accumulation dans l'*API OpenGL* qui permet de mélanger les résultats d'un nombre donné de rendus successifs, les performances sont très faibles lorsque l'on désire obtenir un flou visuellement acceptable. Moins il y a de vues, plus ces dernières seront visibles dans l'image finale, ce qui par conséquent donne moins l'impression d'un flou mais plutôt d'une superposition d'images. Un résultat, produit avec au minimum trente vues, permet de créer un flou correct, mais nécessite un temps de calcul important (cadence de rendu inférieure à 5 secondes). La principale raison est que le *buffer* d'accumulation n'a pas vraiment fait l'objet d'investissements de la part des constructeurs sur les cartes graphiques grand public, ce qui se concrétise par une implantation uniquement logicielle et non matérielle (pas d'espace mémoire dédié au traitement). Toutefois, il serait maintenant possible d'obtenir un résultat assez similaire en utilisant les FBO couplés aux MRT.

Lancer de rayons distribué

Suite aux méthodes de lancer de rayons permettant de générer du flou [MvL00, GA93] moyennant un temps de calcul très élevé, Cook, Porter et Carpenter [CPC84] (figure 42) ont proposés une amélioration nommée «lancer de rayons distribué». Leur modèle de simulation du flou est basé sur le modèle physique et tient donc compte de la présence d'une lentille.

Le principe se calque sur celui utilisé en lancer de rayons pour corriger le problème d'*aliasing*. D'une manière générale l'algorithme fonctionne comme n'importe quel lancer de rayons, si ce n'est que d'une part chaque rayon est dévié à cause de la présence d'une lentille. D'autre part lorsque qu'un rayon atteint le plan de netteté plusieurs autres rayons

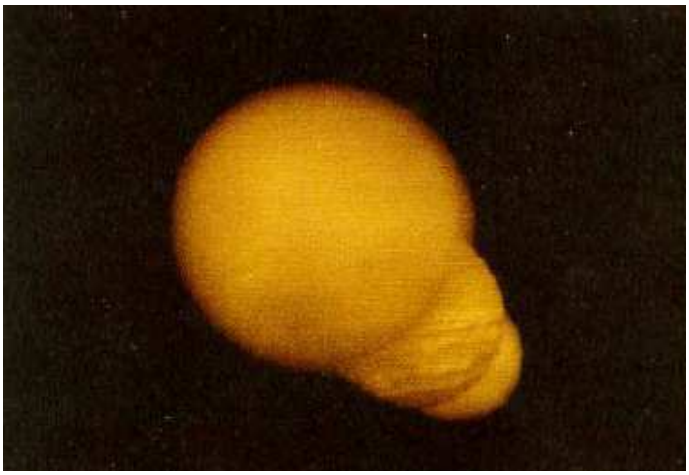


FIG. 42: Illustration de la méthode de génération de flou de profondeur de champ par lancer de rayons distribué *Copyright : cook et al.*

sont générés depuis la surface de la lentille en direction du point d'intersection. De cette manière si un objet est situé autour de la zone de netteté, alors tous les rayons intersecteront la même surface sur un espace réduit. Si au contraire l'objet est éloigné de cette zone alors les rayons fourniront des informations sur différentes surfaces dont les valeurs chromatiques seront combinées pour définir la couleur finale du pixel. Cet effet est illustré dans la figure 43.

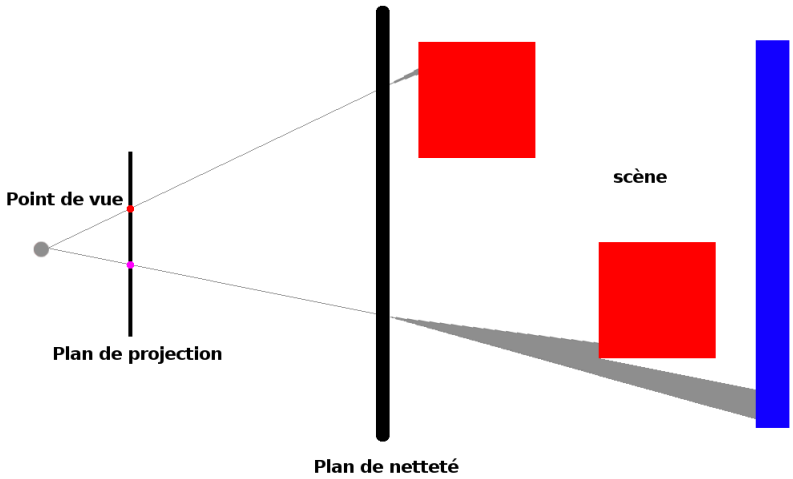


FIG. 43: Fonctionnement de la génération de flou par la méthode de lancer de rayons distribué en fonction de la distance du plan de netteté par rapport au point de vue

Cependant, cette technique utilise le des lancers de rayons, ce qui signifie que les temps de calculs sont longs. Par contre, le flou généré est visuellement très réaliste.

Forward-mapped Z-buffer ou *scattering*

Le principe général de cette méthode réside dans l'utilisation de *sprites*[10] pour simuler chacun des cercles de confusion [PC82, Dem04]. Selon les dimensions de ces tâches, nous allons attribuer une taille différente à chacun des *sprites*. D'un point de vue global, cette méthode applique la contribution de chaque pixel sur chacun de ses voisins (figure 44).

Dans un premier temps, le rendu de la scène est sauvegardé dans une texture ainsi que la carte de profondeur qui lui est associée. La seconde partie de la méthode consiste à afficher un *sprite* pour chaque pixel du rendu, en définissant leur taille en fonction de sa profondeur et en leur attribuant la couleur du pixel. Pour éviter qu'un *sprite* vienne en occulter un autre, une valeur de transparence (inversement proportionnelle à sa taille) est attribuée à chacun d'eux et en veillant à laisser les petits *sprites* au premier plan. Cette dernière opération vise à conserver la netteté des objets qui se trouvent devant des éléments flous. Enfin une dernière étape est requise pour normaliser les valeurs de transparence afin qu'il n'y ait pas de perte de luminosité.

Les performances de cette technique sont très faibles principalement en raison du grand nombre de *sprites* qu'il est nécessaire de générer. De plus, comme la méthode calcule indirectement la contribution de chaque pixel sur ses voisins, cela implique qu'elle peut difficilement bénéficier d'optimisation matérielle comme les *shaders* qui fonctionnent plutôt de manière inversée.

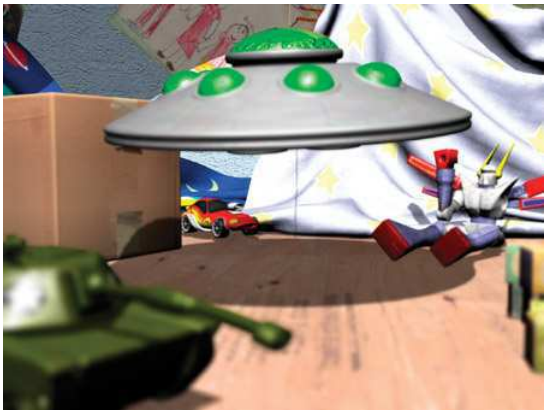


FIG. 44: Illustration de la simulation de flou de profondeur de champ par la méthode *Forward-Mapped Z-Buffer Copyright : J. Demers*

Flou par diffusion simulée

Dans le cadre du développement de ses films d'animation, la société *Pixar* a créé un outil de pré-visualisation des effets de flou pour ses productions. Le rendu final de chaque image nécessitant des calculs très longs, ils ont fait en sorte de développer un algorithme

[10] Vignette image 2D sur lequel est généralement plaqué une texture

qui permet de tester ce genre d'effet en temps interactif [KLO06] en faisant varier les paramètres du système optique.

Cette méthode utilise le principe de diffusion de la chaleur en convertissant les cercles de confusion en diffuseurs de chaleur. De cette manière un pixel va pouvoir influencer ses voisins jusqu'à une distance maximale bien définie dont un résultat est illustré sur la figure 45. La conservation des zones nettes est assurée par l'utilisation même de la formule de diffusion de la chaleur. Un pixel appartenant à un objet net va avoir une température de zéro ce qui implique que la diffusion des possibles sources de chaleurs voisines va s'arrêter à sa proximité.

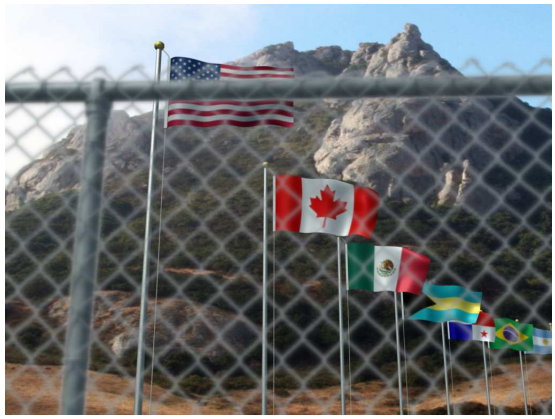


FIG. 45: Flou par simulation de la diffusion de la chaleur *Copyright : Michael Kass, Aaron Lefohn & John Owens*

L'algorithme tire bénéfice des *GPU* en l'utilisant pour résoudre les calculs algébriques associés à l'équation de la chaleur (décomposition LU, résolution d'une matrice tridiagonale,...). Cependant, la complexité même de la méthode ne permet d'atteindre le temps réel que sur des images basse résolution (80 à 90 images par seconde pour une résolution de 256x256) et un temps interactif (7-8 images par seconde) pour une image ayant une résolution de 1024x1024.

3.2.2 Simulation de flou temps réel

Flou par calques

Comme son nom l'indique, le flou par calques est une technique introduite par [Sco94] en 1994 qui se fonde sur une série de plans 2D contenant chacun un objet de l'image initiale. Lors d'une première phase, chaque élément de la scène est rendu dans un claque, puis une opération de *blur* est appliquée sur chacun d'eux et finalement les calques sont combinés pour créer l'image finale.

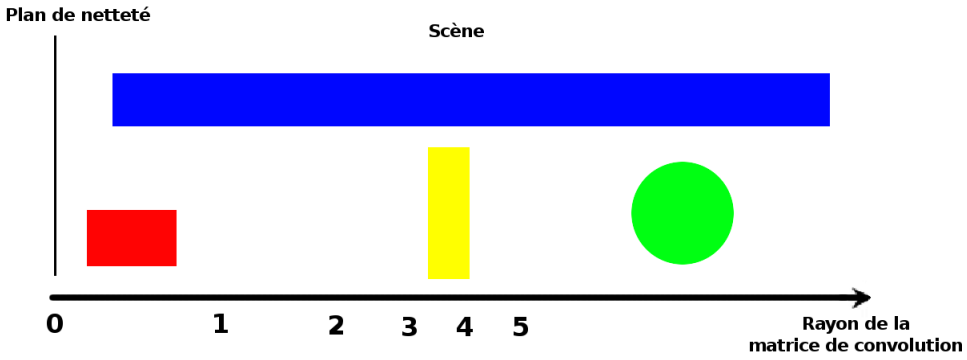


FIG. 46: Présentation de la simulation de flou de profondeur par calques. On remarquera en particulier que l'élément bleu va se voir appliquer uniformement le même flou que le petit objet jaune situé à proximité alors qu'ils ont une taille différente

Le flou est produit en associant une profondeur à chacun des calques selon la distance moyenne de l'objet qu'il contient par rapport au point de vue. Considérant cette information et la position d'un plan de netteté, il est alors possible de définir la quantité de flou à appliquer pour chaque calque. Cette opération est effectuée en utilisant une matrice de convolution de taille variable. La figure 46 illustre le principe de cette technique.

L'intérêt principal de la méthode de flou par calques est qu'elle peut désormais bénéficier des avantages offerts par les *GPU*. Typiquement, l'opération de floutage par une matrice de convolution peut se faire via un programme sur les pixels. Comme la convolution n'est pas une opération qui nécessite de connaître le résultat de ses voisins et qui ne va pas non plus les influencer, il est alors vident qu'une architecture parallèle, telle que celle des *GPU*, est tout à fait appropriée. Enfin, la phase de fusion de l'ensemble des calques qui permet de créer l'image finale peut également bénéficier des mêmes avantages.

Cependant, le résultat visuel va être d'assez basse qualité dans le sens où le flou va être uniforme sur chacun des calques. Le meilleur exemple est de considérer un sol sur lequel sont posés de multiples objets. Un flou va être appliqué sur ce dernier, mais va visuellement entrer en conflit, car pour une même profondeur donnée le sol et l'objet posé dessus n'auront pas la même intensité. Cette limite est illustrée sur la figure 47.

FIG. 47: Illustration de la simulation de flou de profondeur de champ par la méthode des calques *Copyright : J. Demers*

Flou par *mipmapping*

Le *Reverse-mapped z-buffer* est une méthode proposée par [CCAD01] en 2001 qui fonctionne un peu à l'inverse de la précédente tout en conservant une notion de plans. De fait, le résultat du rendu de la scène est récupéré sous plusieurs résolutions en utilisant par exemple la technique de *mipmapping*[11]. Cet ensemble d'images est en fait un précalcul de différents niveaux de *blur*car à chaque étape du *mipmapping*, une moyenne sur les pixels est faîte ce qui revient implicitement à appliquer une matrice de convolution. En redonnant leur résolution initiale, ces images vont alors afficher différents niveaux de flou. Un exemple d'images obtenues par *mipmapping* est présenté dans la figure 48.

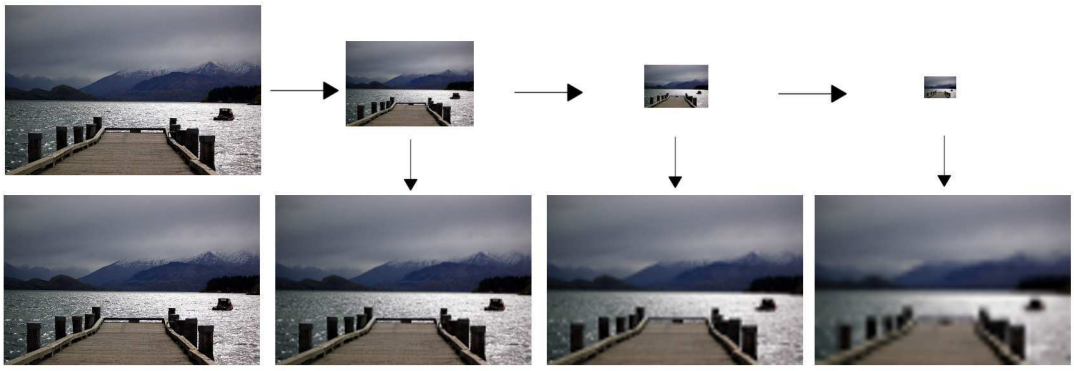


FIG. 48: Précalcul de flous uniformes en utilisant la technique du *mipmapping*. On retrouve sur la première rangée les diffèrentes diminutions de résolution obtenues par *mipmapping*. La seconde rangée illustre l'obtention du flou remettant les images à la résolution originale

[11] Technique permettant, à partir d'une image haute résolution, d'obtenir une série d'images dont les dimensions sont divisées par deux à chaque étape .

L'étape suivante consiste à répartir une série de plans centrés le long de l'axe optique et perpendiculairement à ce dernier. Les dimensions de chacun des plans varient de manière à ce qu'ils soient à la taille de la pyramide de vision 49. Sur le plan le plus proche de la position théorique du plan de netteté, on va placer l'image de la scène avec la plus haute résolution. Pour les autres plans, plus ils seront éloignés de ce dernier, plus l'image qui sera plaquée dessus sera de basse résolution. Finalement une comparaison entre la carte de profondeur de la scène originale et la profondeur de chacun des plans va permettre de sélectionner l'image à associer à chaque pixel.

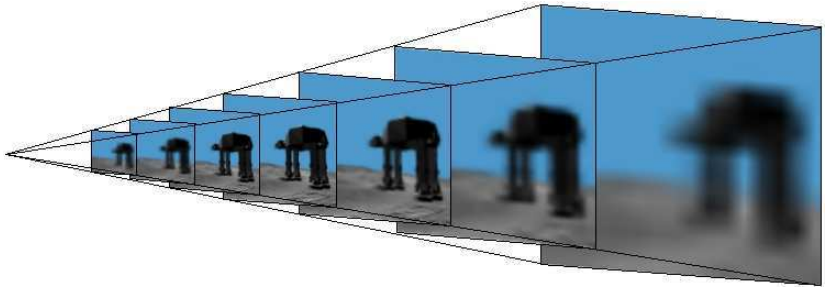


FIG. 49: Positionnement et adaptation des différents plans dans le volume de la pyramide de vision

Concrètement, pour déterminer l'image d'un plan à associer avec un pixel, il suffit de dessiner les plans en partant du plus éloigné et en finissant par le plus proche. La sélection est assurée par la modification du test de profondeur. Les fragments vont être comparés à la carte de profondeur enregistrée dans la première passe et ainsi tant qu'un fragment aura une profondeur inférieure à la valeur sauvegardée, il sera affiché et un parcours de l'arrière vers l'avant assure que ce sera le fragment le plus proche qui sera conservé. L'algorithme suivant résume ce principe :

Algorithm 3.2.1: MIPMAPPING()

Activer un *FBO*
Rendre la scène dans une texture en activant le *mipmapping*
Créer et définir les plans et la position du plan de netteté
Désactiver l'écriture dans le *buffer* de profondeur
Redéfinir la fonction de comparaison des fragments lors du test de profondeur
Dessiner les plans de l'arrière vers l'avant

La méthode offre une manière simple et rapide de produire du flou en temps réel. Cependant, cette efficacité a pour conséquence une diminution de la qualité visuelle (figure 50). Selon la résolution de l'image de départ, le nombre de plans va être assez variable ce qui va limiter les variations dans l'intensité du flou. D'une manière générale, c'est la maîtrise du flou, à proprement parler, qui est délicate dans le sens où il peut être assez

difficile de positionner les plans dans l'espace pour obtenir un résultat visuel particulier ou alors une variation de flou plus souple.

FIG. 50: Exemple de résultat de flou obtenu par la méthode de mipmapping

3.2.3 Applications en réalité virtuelle

Le flou est principalement utilisé lorsqu'il y a besoin de faire un rendu photo-réaliste ou alors tout simplement lorsque l'on désire obtenir les effets induits par un système optique pour mettre en avant un objet dans une scène de synthèse. Les techniques utilisées ne sont généralement pas en temps réel car la qualité visuelle prévaut alors sur la rapidité. On retrouve généralement ce contexte de création dans des œuvres cinématographiques telles que celles proposées par *Pixar* ou *Dreamworks*.

Dans d'autres domaines, le flou peut être utilisé, comme dans les jeux vidéo par exemple. Dans ce cas, il faut alors obtenir des résultats en temps réel, mais la qualité du rendu n'est pas toujours nécessaire. Le flou n'y est alors pas utilisé de manière artistique, c'est-à-dire en utilisant par exemple du flou de profondeur de champ, mais plutôt en appliquant un *blur* sur certaines zones de l'image. De cette manière, il est possible de mettre en avant certains objets de la scène ou alors de souligner certains évènements (blessures, vitesse, focus,...).

Dans le cadre de la réalité virtuelle, Rokita [Rok96] propose en 1996 qu'apparaisse dans de futures applications la simulation de différents effets permettant de mieux percevoir l'univers virtuel d'un point de vue spatial et réaliste. Il oriente ses travaux plus particulièrement sur la simulation du phénomène d'accommodation de notre œil qui se concrétise par l'apparition d'un flou. Son objectif serait alors de l'appliquer à un environnement virtuel en effectuant un *tracking* de l'œil pour positionner le plan de netteté. Au final Rotika ne concrétise que la génération d'un flou avec une très faible intensité pour obtenir des

résultats en temps réel. Suite à cela, Bastian *et al* [BvdHHV00] proposent d'ajouter à leur système *CAVE* une modélisation des effets que peut provoquer l'ajout d'une lentille sur la visualisation. L'observateur peut alors directement modifier les paramètres du système optique pour évaluer leur influence sur la perception de l'environnement. Outre les effets de distorsion, le système est aussi capable de générer du flou Gaussien via la méthode des calques. Un système de suivi de l'utilisateur placé sur ses lunettes permet d'adapter la position du flou dans la scène virtuelle en fonction de la direction de sa tête. Toutefois, cette application se retrouve limitée par les effets de la technique utilisée qui, en plus de nécessiter beaucoup de traitements, fait apparaître de nombreuses incohérences visuelles dans la scène.

Dans le cadre d'une recherche sur les solutions possibles pour limiter les effets de la fatigue oculaire en vision stéréoscopique, Arnaldi *et al* [AFT03] proposent l'emploi d'un flou adaptatif. L'image commence par être décomposée par traitement en ondelettes se basant sur la disparité locale pour être ensuite recomposée uniquement dans les parties ne correspondant pas aux hautes fréquences. De cette manière, les zones difficiles à reconstruire sont floutées, ce qui va en principe, inciter l'utilisateur à diriger son regard vers les parties nettes. Cependant, selon les contextes, les calculs nécessaires ne permettent pas d'utiliser cette solution en temps réel.

Hillaire *et al* [HLCC07] proposent d'utiliser du flou en environnement virtuel dans l'objectif d'aider à la perception de l'environnement et de rendre les images de synthèse moins parfaites. Deux types de flous sont utilisés et combinés. Le premier, appelé flou de périphérie, va s'appliquer au fur et à mesure que l'on va s'éloigner du centre de l'image. Le second flou est le flou de profondeur qui va automatiquement être paramétré en fonction du contenu d'une zone d'intérêt (généralement le centre de l'image). La mise en application se fait sur un jeu du type tir à la première personne avec lequel une expérimentation est menée afin de connaître l'impact du flou sur la jouabilité.

Les premiers travaux de Bastian *et al* [BvdHHV00] sont principalement destinés à reproduire des effets d'optique en interaction avec l'utilisateur. Cette approche, orienté uniquement rendu, pourrait donc être utilisée dans un tout autre contexte que la réalité virtuelle car elle ne propose pas, par exemple, d'améliorer l'immersion de l'utilisateur dans l'environnement virtuel. Au contraire, Arnaldi *et al* [AFT03] proposent d'utiliser le flou de façon à réduire la fatigue oculaire provoquée par les images stéréoscopiques. Cependant, leur solution n'est réalisable qu'à la condition de disposer d'un grand nombre de ressources de calcul pour pouvoir être utilisée en temps réel. Enfin, la méthode de Hillaire *et al* [HLCC07] s'inscrit dans la lignée de nos travaux, bien que soumis après les nôtres.

3.3 Simulation de flou sur images stéréoscopiques pour la réalité virtuelle

Notre objectif est de proposer une utilisation du flou en réalité virtuelle visant à entretenir, voire améliorer, le sentiment de présence. Tout d'abord, nous voulons éviter que l'utilisateur ait des fatigues oculaires ou des maux de tête causés lorsque la fusions d'une image stéréoscopique est trop difficile. En appliquant le flou sur des zones spécifiques de l'image, nous pouvons faire en sorte de maintenir l'effet d'immersion de l'utilisateur. Ensuite, lors de la phase de découverte d'un environnement virtuel, l'utilisateur peut se sentir perdu à cause de la quantité de nouvelles informations affichées. Nous proposons donc d'utiliser le flou pour réduire cette perte de point de repère. Ainsi, en rendant net certains objets de la scène, nous allons créer une interaction entre l'univers virtuel et l'utilisateur.

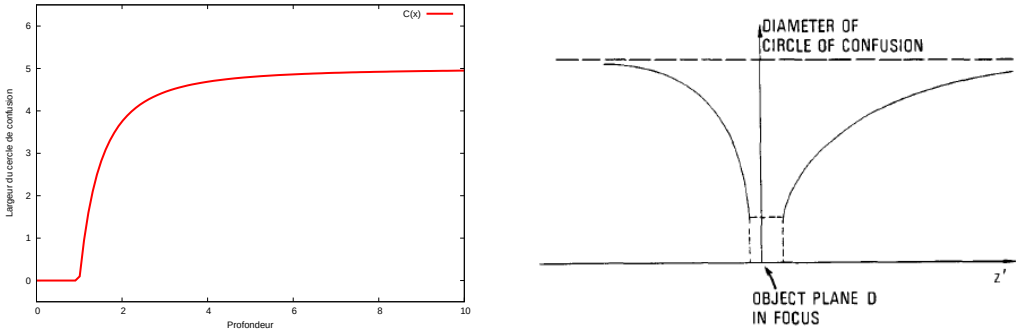
Notre volonté n'est pas de simuler du flou pour reproduire le réalisme de la perception de l'œil humain ou la recherche d'un effet artistique, mais de l'utiliser pour masquer les zones de l'image que nous considérons comme secondaires et/ou difficiles à fusionner. Le choix de l'utilisation du *blur* pour arriver à cette fin semble être une solution appropriée à cause de sa simplicité de mise en œuvre. Cependant, il apparaît clairement que visuellement le flou sera uniforme à cause de l'application de la matrice de convolution à taille unique. La simulation d'un flou de profondeur de champ se révèle alors plus naturelle tout d'abord parce que c'est celui que nous observons le plus souvent et ensuite parce que c'est celui qui a la meilleure esthétique. Mais son application est toutefois un peu plus compliquée que celle du *blur*. Notre objectif consiste donc à fonder notre solution sur une méthode tenant compte des avantages de chacun des flous.

Notre méthode a pour objectif de calculer le cercle de confusion pour chacun des pixels de l'image, en tenant compte de la profondeur de l'objet auquel il est associé. Nous proposons donc une simplification du modèle proposé par Potmesil et Chakravarty [PC82] qui nous permette plus de flexibilité en soustrayant les paramètres liés aux caractéristiques physiques du système optique. Ainsi, les notions d'ouverture et de distance focale sont remplacées par la fonction empirique suivante qui utilise la distance de l'objet traité par rapport au plan de netteté :

$$C(x) = \begin{cases} (-\frac{1}{x^{\beta}+1-\frac{l}{2}} + 1.0) \times \alpha & \text{si } x > \frac{l}{2} \\ 0 & \text{sinon} \end{cases} \tag{3.1}$$

avec x la distance au plan de netteté et $\alpha > 1$, β est un coefficient permettant de faire varier la pente de la courbe, l la largeur de la zone de «profondeur de champ» et α un coefficient de mise à l'échelle de la courbe. Comme nous ne cherchons pas à obtenir un flou réaliste (la courbe n'est pas la même selon que l'on se trouve devant ou derrière le plan de netteté), l'approximation de la courbe illustrée sur la figure 51 convient pour le modèle que nous voulons mettre en œuvre. Ainsi, la courbe d'évolution du cercle de

confusion sera identique à l'avant et à l'arrière du plan de netteté.



(a) (b)

FIG. 51: Comparaison des deux courbes d'évolution de la taille du cercle de confusion en fonction de la distance au plan de netteté : (a) notre modèle ($\alpha = 5$, $\beta = 2$) et (b) le modèle de Potmesil et Chakravarty *Copyright : [PC82]*

La mise en pratique du concept de cercle de confusion se fait en utilisant le principe de création d'un *blur*, c'est-à-dire en appliquant une matrice de convolution. La dimension de cette dernière est définie par la fonction 3.1 dont on va récupérer les parties entières et qui détermine le «rayon» de la matrice de convolution. Ainsi la taille du cercle de confusion va augmenter très rapidement lorsque la distance par rapport au plan de netteté commence à dépasser celle de la profondeur de champ, puis va progressivement converger vers une taille maximale. La coefficient de mise à l'échelle α de notre fonction permet de définir cette taille maximale du cercle de confusion selon le flou que l'on désire obtenir.

Le contenu de la matrice de convolution a aussi son importance car il influencera l'aspect visuel du flou. Tout naturellement la forme s'oriente vers un cercle pour représenter au mieux l'aspect que prend la tâche de confusion. De plus nous privilégions l'utilisation d'une répartition des valeurs sous la forme d'une gaussienne ce qui permet de donner moins d'importance aux valeurs à la périphérie du cercle de confusion et ainsi appliquer une atténuation en fonction de la distance. Les composantes de ce filtre sont calculées à l'aide de l'équation 3.2 avec une déviation standard $\sigma \geq 1$.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{\frac{-x^2+y^2}{2\sigma^2}} \tag{3.2}$$

L'utilisation des *shaders* se révèle avantageuse lorsqu'un algorithme nécessite un tel traitement sur les pixels. De cette manière, l'algorithme bénéficie d'une accélération matérielle via la parallélisation des calculs. Cependant, il n'est pas possible, lors de la passe de rendu, d'avoir accès dans le *fragment shaders* aux pixels voisins ce qui ne permet pas d'appliquer la matrice de convolution directement. La solution consiste alors à appliquer l'algorithme en deux passes. La première passe se charge de rendre la scène dans

une texture au moyen par exemple de l'extension *OpenGL Frame Buffer Object* et par la même occasion, une carte de profondeur est sauvegardée. Enfin, la seconde passe applique le filtre de flou à partir de la texture précédemment calculée en fonction des valeurs de la carte de profondeur. L'algorithme peut alors être décrit de la manière suivante :

Algorithm 3.3.1: RENDU DU FLOU()

for each $affichage$
do {
  texture $S \leftarrow rendudelascne$
  texture $P \leftarrow$ carte de profondeur
  Appliquer le pixel *shader*
  for each *pixel*
  do {
    $profondeur \leftarrow P$
    $d \leftarrow \mid n - p \mid$
    comment: Calcul de la taille du cercle de confusion en fonction de d
    $conf \leftarrow C(d)$
    $pixel \leftarrow 0$
    for each $x, y \in conf$
      do $pixel \leftarrow pixel + (S(x, y) \times G(x, y))$

On peut noter qu'en ce qui concerne la carte de profondeur, nous ne sauvegardons pas le *buffer* de profondeur d'*OpenGL* car ce dernier est le résultat d'une projection et d'une mise à l'échelle dont les valeurs doivent être retransformées pour obtenir les données de profondeur réelles. Notre choix est donc de créer la carte de profondeur lors de la première passe en utilisant le programme sur les sommets. Il y est possible de multiplier chaque sommet par la matrice *modelview* afin d'obtenir les coordonnées dans le repère d'œil puis de transmettre sa distance par rapport au point de vue du programme sur les pixels qui le sauvegardera dans une texture. Les programmes des *shaders* en *GLSL*, pour la première et la seconde passe, sont présentés dans les codes 3.1 et 3.2.

Listing 3.1: Code source du *shader* de la première passe

```
1  //****************
2  //Vertex Shader
3  //****************
4  varying out float depth;
5
6  void main(void)
7  {
8      ... // Code traditionnel
9      depth = length((gl_ModelViewMatrix * gl_Vertex).xyz);
10     gl_Position = ftransform();
11 }
12
13 //****************
```

```
14 //Fragment Shader
15 //****************
16 varying in float depth;
17
18 void main(void)
19 {
20   vec4 color = gl_Color;
21    ... // Code traditionnel (calcul de l'illumination par exemple)
22   gl_FragData[0] = color;
23   gl_FragData[1] = vec4(depth);
24 }
```

Listing 3.2: Code source du *shader* de la seconde passe

```
1
2 //****************
3 //Fragment Shader
4 //****************
5 uniform vec2 kernelsize;
6 uniform float kernel[kernelsize.x * kernelsize.y];
7
8 uniform sampler2D image;
9 uniform sampler2D depth;
10
11 uniform float dof;
12 uniform float beta;
13 uniform float alpha;
14 uniform float focusdepth;
15
16 void main(void)
17 {
18    vec2 offset = vec2(1.0)/kernelsize;
19    float depth = texture2D(depth, gl_TexCoord[0].st).r;
20    depth = abs(depth - focusdepth);
21    int coc = 0;
22    if(depth>dof/2.0)
23      coc = floor((-1/(pow(depth,beta)+1-dof/2.0)+1.0)*alpha);
24
25        vec4 color = vec4(0.0);
26        vec2 ref =(vec2(kernelsize.x, kernelsize.y)+1)/2;
27
28    for(int i = -coc; i<=coc; ++i){
29       for(int j = -coc; j<=coc; ++j){
30          color += kernel[kernelsize.x*(j + ref.y) +  i + ref.x]*
31          texture2D(image, gl_TexCoord[0].st+offset*vec2(i,j)).r;
32      }
33    }
```

```
34      gl_FragColor = color;
35  }
```

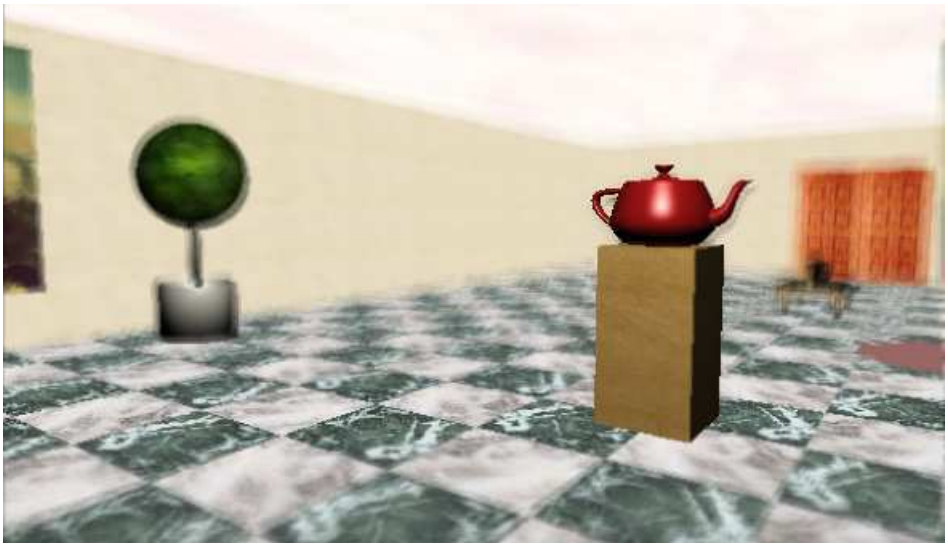


FIG. 52: Résultat obtenu par l'application de notre méthode de simultation de flou

La figure 52 illustre l'utilisation de notre algorithme. Le fait d'utiliser la carte graphique pour créer notre flou de profondeur de champ permet d'obtenir un résultat en temps réel, même si la surface de traitement a été doublée par deux, en raison de l'application aux images stéréoscopiques. Les *shaders* permettent ainsi d'exploiter facilement une architecture parallèle qui se prête très bien à notre méthode de génération du flou. Avec des performances supérieures à 30 images par seconde pour la mise en œuvre de cet effet, notre solution s'adapte parfaitement à une utilisation dans un contexte d'aide à la perception en réalité virtuelle.

3.4 Emploi du flou en réalité virtuelle

L'immersion visuelle peut être améliorée de deux manières. Tout d'abord, en faisant en sorte que les zones de l'image stéréoscopique pouvant provoquer des maux de tête n'attirent pas l'attention de l'utilisateur, ensuite en mettant en avant certains éléments de notre scène ,considérés comme importants, pour que l'observateur puisse focaliser son attention (figure 53). En tenant compte de ces deux besoins, nous proposons une solution utilisant notre méthode de génération et de positionnement du flou dans les environnements virtuels. Il existe de multiples critères pouvant être utilisés pour sélectionner les objets devant apparaitre nets dans la scène.

(a) (b)

FIG. 53: Le positionnement précis du plan de netteté dans une image de synthèse *(a)* va permettre d'attirer l'attention de l'observateur sur un élément bien determiné comparé à une image complétement nette *(b)*

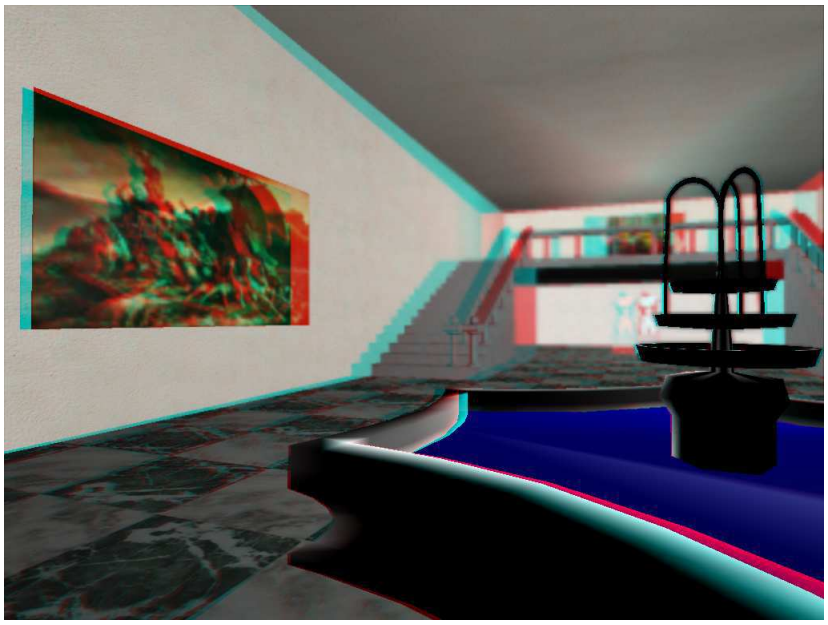


FIG. 54: Le plan de netteté est placé sur l'objet le plus proche du point de vue

3.4.1 Positionnement automatique

Ce processus de positionnement du flou peut se faire de manière automatique en se servant des caractéristiques composants les éléments. Il est ainsi possible de fonder le choix en considérant leur taille, leur couleur, leur nature, leur position à l'écran, *etc.*

Lorsque l'on parle de taille d'un objet, il faut en considérer deux types. D'un côté on va définir la taille en se fondant sur ses dimensions et de l'autre, la taille va être définie comme la surface qu'il occupe à l'écran. Dans le premier cas, les dimensions sont généralement associées à un volume englobant (*bounding box*, *bounding sphere*, ...) qui approxime au mieux l'objet et qui permet d'en évaluer sa taille en déduisant l'espace qu'il

occupe dans la scène. Pour calculer la surface (en pixels) qu'occupe un objet à l'écran, une solution consiste à utiliser l'extension *OpenGL ARB_ occlusion_ query* [Khr03]. En mettant l'application dans un contexte graphique particulier, il est possible de récupérer pour chaque objet le nombre de pixels ou de fragments qui lui sont associés. Toutefois, ce processus ne peut se faire qu'en affichant deux fois les objets dont on désire connaître la taille, ce qui par conséquent peut légèrement réduire les performances.

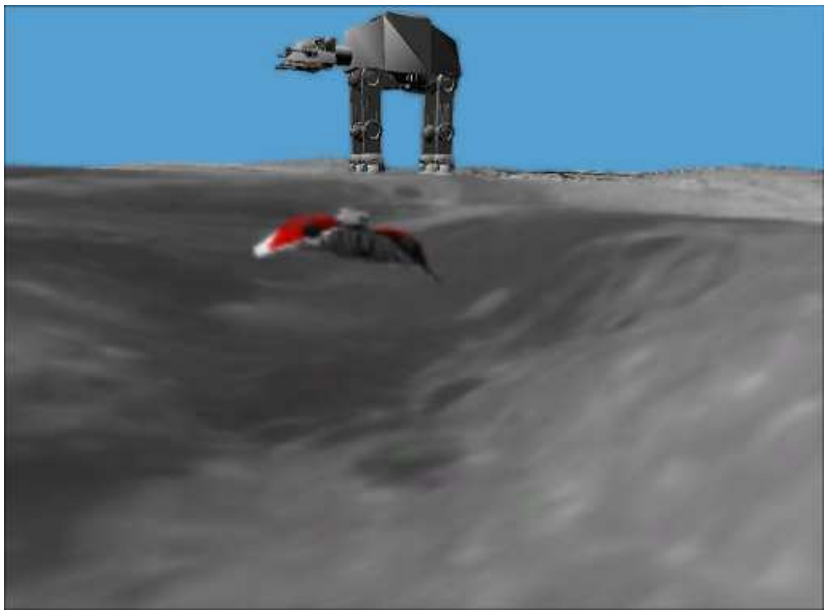


FIG. 55: Le plan de netteté est positionné sur l'objet (hors terrain) occupant le plus de place à l'écran

Le critère de sélection de positionnement du plan de netteté ne peut pas être fixé de manière uniforme pour l'ensemble des applications, mais va plutôt nécessiter d'être configuré en fonction du contexte d'utilisation. Les besoins ne vont pas être les mêmes lors de l'exécution d'une expérience à des fins ludiques ou lors d'une simulation. Dans le premier cas, avec par exemple un jeu à la première personne, il est nécessaire que l'attention de l'utilisateur se porte sur les objets mobiles dont la taille est relativement importante à l'écran. Dans ce genre de condition d'utilisation, le regard de l'utilisateur a une forte tendance à se concentrer sur le centre de l'écran et donc mettre des adversaires en avant en employant du flou sur les zones de moindre intérêt. Il va alors avoir pour conséquence d'accroître sa perception.

Une autre approche de l'utilisation du flou est de l'appliquer à un cas de téléprésence qui implique la participation de plusieurs intervenants. Lors de ces réunions virtuelles il est toujours difficile de savoir qui mène la discussion car les détails ne sont pas forcement simples à percevoir. En appliquant une zone de netteté sur l'interlocuteur principal, il devient possible d'aider l'utilisateur à focaliser plus facilement son attention. De la même façon, le flou peut être appliqué pour mettre en avant un orateur lors d'une téléconférence.

Outre le positionnement du plan de netteté, c'est aussi la forme du flou qui a son importance. Un utilisateur, libre de ses mouvements dans un environnement virtuel qu'il ne connaît pas, a besoin de trouver des points de repère. Utiliser un flou trop important dans ces conditions peut donc entraîner une diminution de la perception. La solution consiste, lors de cette phase de découverte, à utiliser le flou de faible intensité couplé à une grande profondeur de champ de manière à ce qu'il ne soit appliqué qu'aux zones des images stéréoscopiques qui peuvent provoquer des maux de tête.

3.4.2 Positionnement contrôlé

Une autre manière d'aborder le positionnement et la définition de la zone de netteté dans un environnement simulé, consiste à se placer dans le contexte d'une visite virtuelle. Dans de telles conditions, l'observateur a généralement en face de lui une grande variété d'œuvres dont il attribue une importance égale ou dirigée selon ses propres critères de préférence. Lors d'un simple déambulement, le flou est appliqué pour simplement éviter à l'utilisateur de se concentrer sur les zones de l'image stéréoscopique trop difficile à reconstruire. Ces zones sont définies en fonction de la distance inter-oculaire, du plan de convergence des images stéréoscopiques et des dimensions de la surface sur laquelle les images sont diffusées.

En estimant de manière empirique que l'écart maximal entre les deux images ne doit pas excéder une certaine distance σ. On peut en déduire que les zones de relief et de profondeur, pouvant provoquer des maux de tête, sont respectivement situées à une distance δ_r et δ_p du point de vue (figure 56). A l'aide des relations géométriques des images stéréoscopiques, nous pouvons les définir de la manière suivante :

$$\begin{array}{ll} \rho = & \frac{\sigma \times W}{W_{screen}} \\ \delta_r = & \frac{dio \times f}{\rho} \\ \delta_p = & \frac{f \times \rho}{\rho - dio} \end{array} \tag{3.3}$$

avec dio la distance inter-oculaire, W la largeur du plan de convergence, W_{screen} la largeur de l'image projetée sur l'écran et f la distance du plan de convergence depuis le point de vue. Notons que la distance σ peut être différente pour δ_r et δ_p.

À proximité des œuvres, on va naturellement diriger la zone de netteté dessus afin que l'attention de l'observateur s'y concentre. Les dimensions de cette zone vont être automatiquement adaptées en fonction de l'objet à laquelle elle est associée. Pour mettre en place ce processus des «aires de déclenchement» sont prédéfinies à proximité des éléments d'intérêt. Il est également possible de déterminer la visibilité de ces aires pour aider l'observateur à les trouver. L'autre bénéfice de ces aires est qu'il est également possible de déclencher des événements autres que le positionnement du flou comme un commentaire

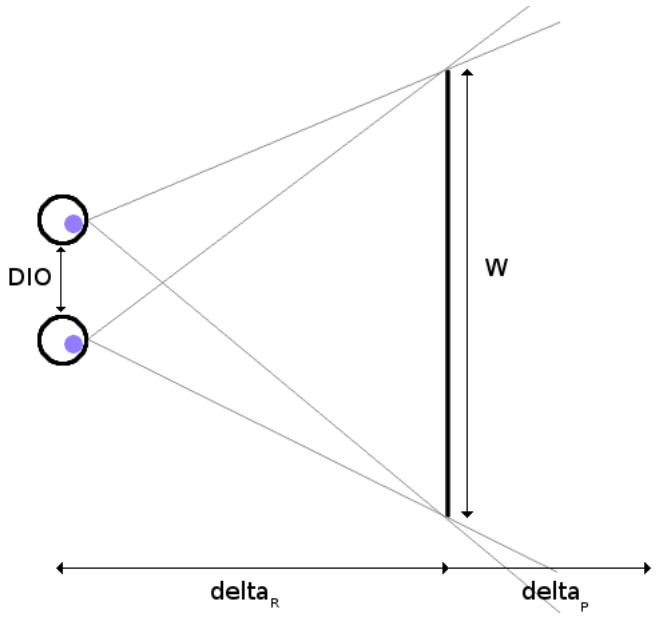


FIG. 56: Illustration des zones considérées comme ne provoquant pas de fatigue oculaire. Un flou est appliqué sur les autres

audio lors de la visite virtuelle.

Dans ce cas, l'avantage d'utiliser le flou de profondeur de champ plutôt que du *blur* est qu'il permet de conserver une visibilité à proximité de l'objet d'intérêt de manière suffisamment souple pour que cela ne «choque» pas l'observateur.

Lorsque l'on parle de positionnement contrôlé, il est aussi possible de penser à l'utilisation d'un suivi du regard. En utilisant une caméra dirigée vers l'utilisateur, un algorithme de type *gaze tracking* [JZ02, MMM06, ZJ05] et un calibrage, il devient possible de connaître la direction vers laquelle se porte le regard de l'utilisateur. On peut alors en déduire l'objet qu'il est en train d'observer. Il serait imaginable d'utiliser ce principe afin de positionner le plan de netteté en cherchant à l'associer à l'objet sur lequel l'utilisateur porte son attention. Si nous suivons ce raisonnement, cela revient à simuler la vision humaine en ajoutant du flou à celui que nous créerons naturellement. Si le suivi du regard est utilisé, il doit servir à définir une zone de l'image dans lequel on va positionner le plan de netteté de manière automatique.

3.4.3 Positionnement scripté

La dernière catégorie de positionnement de flou consiste à définir le placement du plan de netteté en suivant un script de déroulement préalablement écrit. Une telle utilisation est alors très analogue à celle que l'on peut trouver dans le domaine du cinéma avec la mise en avant d'objets ou de personnes. Ainsi, à l'image d'un metteur en scène, nous

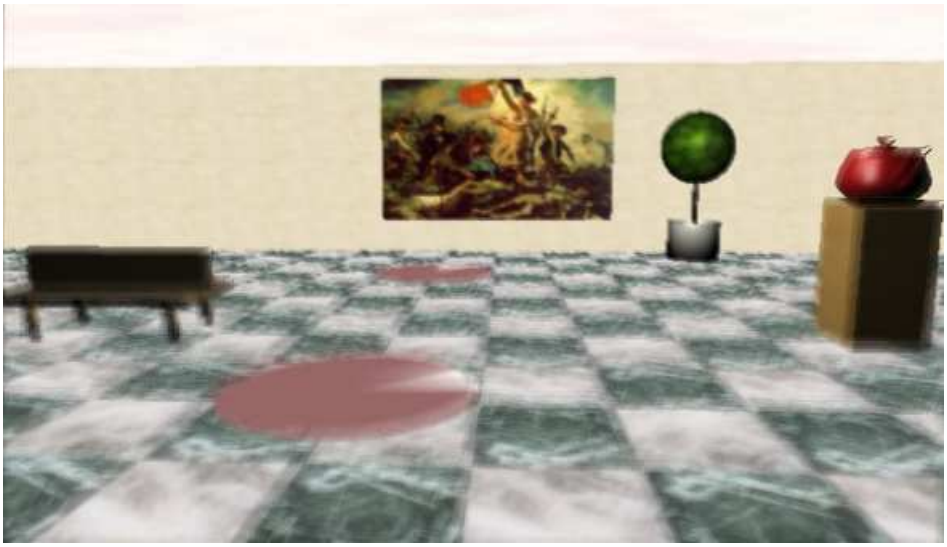


FIG. 57: En se positionnant sur des pastilles reparties sur le sol du musée virtuelle, l'utilisateur va déclencher un positionnement du flou sur une une œuvre bien précise

allons pouvoir définir les points d'attention lors d'une animation.

Pour que cela soit possible nous avons ajouté au système d'animation du contenu de notre scène la prise en compte du positionnement du plan de netteté. Le listing 3.3 présente un exemple de script verbeux permettant d'animer les éléments avec le placement du plan de netteté. La scène, dont le résultat est illustré dans le figure 58, est constituée de deux objets. Le plan de netteté est tout d'abord positionné sur l'objet immobile, puis est placé sur le vaisseau une fois qu'il est à une distance suffisante pour bien marquer sa présence et la menace qu'il représente. Ce dernier se met à tirer sur l'objet immobile et l'attention est donc portée vers les projectiles afin de voir s'ils vont atteindre ou pas leur cible.

Listing 3.3: Exemple de script d'animation dans une version simplifiée

```
1  temps 0
2  // definition des parametres du robot
3  positionne objet 1 vecteur [30.0,30.0,-30.0]
4  affiche objet 1
5  nettete objet 1
6  stoppe objet 1
7  // definition des parametres du vaisseau
8  trajectoire objet 0 courbe 0
9  vitesse objet 0 reel 100.0
10 bouge objet 0
11 affiche objet 0
12 // definition des parametres du projectile
13 positionne objet 2 vecteur [0.0,0.0,0.0]
```

```
14 vitesse objet 2 reel 1000.0
15 stoppe objet 2
16 enleve objet 2
17
18 temps 20000
19 // Le vaisseau tire le projectile
20 // La trajectoire et la position d'origine du projectile sont definies d
21 demarre action 1 objet 0
22 affiche objet 2
23 bouge objet 2
24
25 temps 30000
26 nettete objet 2
27
28 temps 50000
29 nettete objet 0
30
31 temps 80000
32 // Le vaisseau retire le projectile
33 demarre action 1 objet 0
34
35 temps 100000
36 redemarre
```

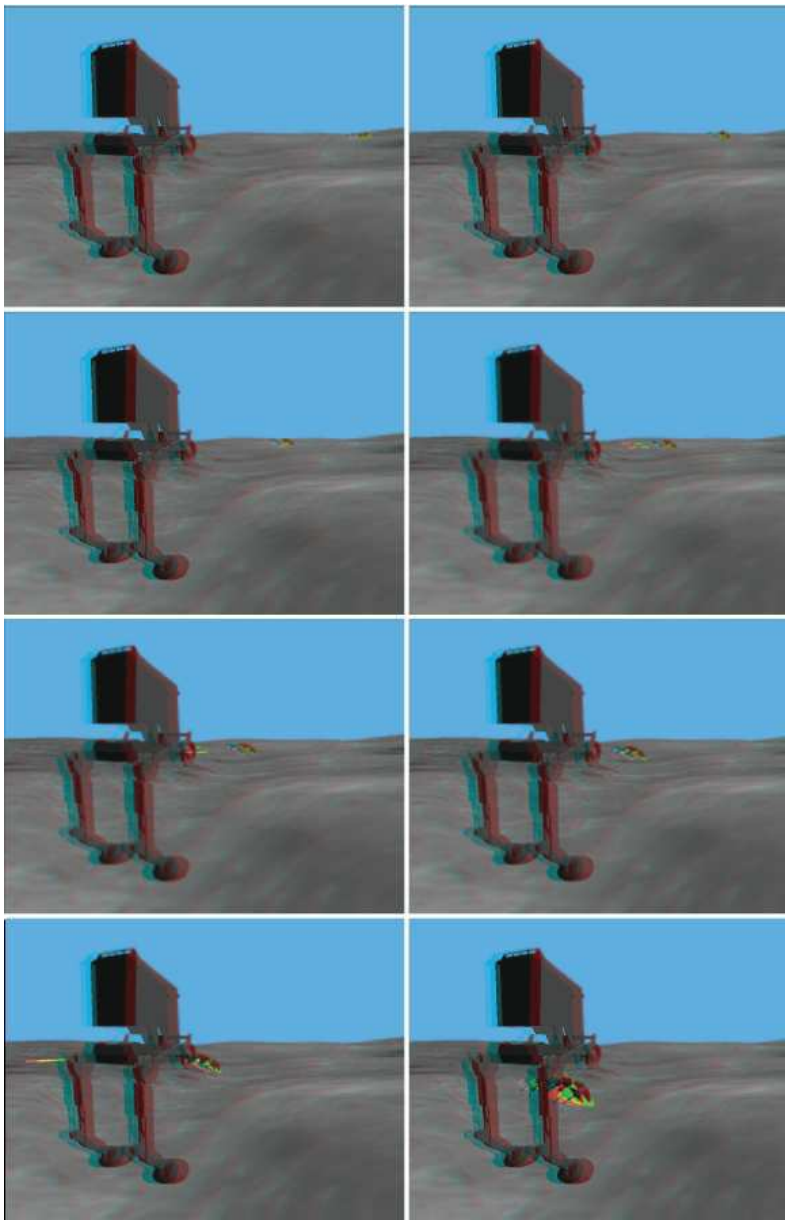


FIG. 58: Le plan de netteté est positionné sur l'objet (hors terrain) occupant le plus de place à l'écran

Troisième partie

Présence de soi dans l'environnement virtuel

1

Se percevoir en réalité virtuelle

1.1 Définition

L'obtention du sentiment de présence est, comme cela a déjà été évoqué, l'objectif ultime d'une expérience de réalité virtuelle. Il permet à un utilisateur d'avoir ainsi l'impression d'appartenir à l'environnement factice qui lui est présenté. Cela passe par la nécessité d'utiliser les quatre piliers que nous avons défini comme étant l'immersion, l'interaction, le maintient de la boucle action-perception et la création d'émotions. Cet ensemble va amener l'utilisateur à avoir le sentiment d'appartenir à l'univers qui lui est présenté car ce dernier va avoir une influence sur ses perceptions mais également parce que les actions de l'utilisateur vont avoir un impact sur les éléments qui composent cet univers.

Lorsqu'on aborde le sujet du sentiment de présence, on peut se demander quels sont les indices qui participent à faire croire à l'utilisateur qu'il est acteur d'un environnement virtuel. En fait, une solution consiste à lui donner l'impression qu'il peut se projeter dans cet espace en faisant par exemple en sorte de s'y apercevoir indirectement comme le souligne Slater *et al.* [SU93]. Ce principe de faire croire à notre présence a déjà été exploité dans le jeu vidéo comme dans *DOOM III* où des miroirs sont disséminés un peu partout dans les niveaux et permettent d'observer la représentation de notre personnage. Toutefois, dans le cas de la réalité virtuelle nous ne désirons pas faire endosser un rôle à l'utilisateur, mais bien faire en sorte que ce soit sa propre personne qui soit visible dans le monde virtuel.

1.2 Avatar et réalité virtuelle

Dans le domaine de réalité virtuelle quelques recherches ont été effectuées pour que nous ayons l'impression d'être intégrés dans un environnement virtuel. Un premier cas d'utilisation est celui de la téléprésence dans lequel l'utilisation d'une représentation de l'utilisateur, ou avatar, est intégré au monde virtuel afin de faciliter les interactions. Dans ces situations, l'avatar a un rôle d'interface entre l'univers et l'utilisateur via lequel ce dernier va également essayer de s'identifier. Pour cela une phase initiale est généralement

mise en place pour que l'utilisateur puisse définir la manière selon laquelle il sera perçu par les autres participants. Finalement, ce n'est pas forcement une représentation réaliste qu'il va donner de lui même ; ce qui ne permet pas à engendrer le sentiment d'être présent dans l'univers virtuel mais reste plutôt au niveau de la manipulation d'un personnage. Dans ces conditions, une notion de distance persiste.

Une autre utilisation possible des avatars en réalité virtuelle, consiste à reconstruire le modèle de l'utilisateur en vue de l'injecter dans l'environnement virtuel en tant que simple copie. Dans ces conditions l'avatar devient un nouvel acteur qui va certes réagir en effectuant les mêmes mouvements que son modèle originel, mais n'apparaît cependant pas comme le propre prolongement de l'utilisateur. Une nouvelle fois, l'avatar est considéré comme une interface qui va pouvoir agir sur les composantes du monde en faisant par exemple usage d'un moteur physique comme dans [ABF+04]. Nous pouvons également souligner que certaines utilisations de l'avatar sont intentionnellement faites sous forme d'un clone de l'utilisateur comme l'ont proposé Gross *et al.* dans [GWN+03]. Ils justifient cette utilisation d'un point de vue artistique en considérant par exemple que le support d'affichage est un miroir via lequel l'utilisateur peut s'observer. De la même manière, l'avatar est utilisé dans les simulations de sport pour permettre à l'utilisateur d'analyser en temps réel ses propres mouvements. À la différence d'un simple miroir, l'emploi d'un système de réalité virtuelle lui permet, en plus d'avoir la possibilité de s'observer, de pouvoir récupérer un nombre d'informations supplémentaires obtenues par une étude automatique de l'avatar.

Le cas le plus courant de l'exploitation d'un avatar en réalité virtuelle pour représenter l'utilisateur est celui de l'emploi d'un *HMD*. En portant ce dernier il est impossible de voir autre chose que l'environnement virtuel et par conséquent, il n'est pas possible de s'apercevoir. Pour éviter de donner l'impression de «flotter», un avatar est utilisé en lieu et place du réel utilisateur comme dans [UAW+99]. De cette façon ce dernier a l'impression d'observer son propre prolongement dans le virtuel. Cependant, la plupart des solutions s'accompagnent d'interfaces de *tracking* encombrantes qui peuvent venir perturber l'obtention du sentiment de présence.

1.3 Notre point de vue sur la perception de soi dans le virtuel

Dans notre étude, nous allons nous placer dans le cadre d'une utilisation de l'avatar dans un système fondé sur la projection d'images (l'utilisation d'un *HMD* est donc exclue). La plupart des approches visant à intégrer une représentation de soi, dans l'environnement virtuel, le font de manière à ce que l'avatar copie nos moindres mouvements, un peu comme à l'image d'un marionnettiste qui manipule son pantin sur la scène ou d'un mime. Deux cas de figure sur l'utilisation d'un avatar se présentent : il est possible de l'observer dans son intégralité ou uniquement certaines des parties de ce corps virtuel.

L'observation d'une partie d'un avatar, comme les bras ou les jambes, s'inscrit parfaitement dans le cadre d'une utilisation avec un *HMD* car quoiqu'il puisse faire, l'utilisateur a son champ de vision envahie par une image de l'environnement virtuel. Utiliser une telle représentation pour simuler ses gestes aide alors à l'immersion. Toutefois, une contradiction va apparaître lorsque cette même technique est utilisée dans un système fondé sur la projection d'images. Dans une telle situation, l'utilisateur a alors pleinement conscience de son existence dans le sens où il suffit qu'il bouge la tête pour qu'il ait une vision de lui-même. La contradiction va alors venir de l'opposition qu'il y a entre, par exemple, l'observation de son propre bras et de sa copie virtuelle. Ce dédoublement a alors pour effet d'amplifier l'aspect irréel de l'environnement virtuel, ainsi l'utilisateur n'identifiera pas cette image comme son propre prolongement.

De la même manière, l'incrustation d'une copie conforme de l'utilisateur en tant qu'acteur de l'univers virtuel ne va pas aider à créer un sentiment de présence. Cette représentation aura beau effectuer les mêmes gestes, porter les mêmes vêtements ou avoir les mêmes caractéristiques physiques, elle n'en restera pas moins une copie à laquelle on s'identifiera comme on pourrait le faire dans un jeu vidéo à la troisième personne[12]. Autrement dit, dans une telle situation, une distance entre l'utilisateur et sa représentation existe et ne va pas dans le sens du sentiment de présence.

Si un avatar doit être intégré à un environnement virtuel alors elle doit se faire de manière naturelle, c'est-à-dire sans que cela ne vienne contredire notre perception. Pour cela nous proposons que l'avatar ne soit visible que par le biais de surfaces réfléchissantes. De plus l'avatar ne doit pas être une représentation quelconque d'un humanoïde mais bien la copie conforme de l'utilisateur. L'impression de manipulation d'un objet virtuel n'est effectivement pas aussi représentative que de voir un «bras» virtuel le faire ; mais peut être obtenu en employant des images stéréoscopiques qui donnent l'illusion que les éléments sont présents devant nous. Par contre, il est certain que cela ne permet pas d'en simuler la consistance, mais après tout, l'utilisation d'un avatar en tant que prolongement de notre corps ne le fait pas plus.

Notre objectif est donc d'aider l'utilisateur à se projeter dans l'environnement virtuel en commençant par créer sa copie virtuelle en temps réel afin de conserver la boucle action-perception. Une fois l'avatar créé, il est injecté dans la scène d'une manière qui puisse paraître la plus naturelle possible à l'utilisateur comme des miroirs ou des ombres. Cela va alors avoir pour effet d'accroître le sentiment d'immersion de l'utilisateur pouvant même aller jusqu'à produire des émotions. En utilisant ce processus d'avatar, nous comptons ainsi augmenter les possibilités de générer chez l'utilisateur le sentiment de présence.

[12]Type de jeu dans lequel on agit sur un personnage depuis un point de vue légèrement reculé

2

Création d'un avatar : État de l'art

Les méthodes permettant de créer une copie virtuelle d'un sujet réel placé, par exemple, au sein d'un système de réalité virtuelle sont nombreuses et assez variées. Certaines visent à ne produire un rendu qu'à base d'images depuis un point de vue inédit ou alors à directement recréer le modèle géométrique d'une personne ou d'un objet bien précis.

Notre objectif étant d'exploiter l'avatar dans un contexte de réalité virtuelle, il est obligatoire d'obtenir un résultat en temps réel ou au moins interactif afin que la boucle action-perception ne soit pas brisée. Nous ne présenterons donc que les méthodes qui respectent ces conditions. Mais comme elles ont toutes en commun d'utiliser des caméras calibrées, nous commencerons par faire une présentation des méthodes de calibrage.

2.1 Calibrage des caméras

2.1.1 La caméra projective

Pour représenter une caméra, le modèle le plus couramment employé est le sténopé (*pinhole* en anglais) en partie à cause de sa simplicité que l'on retrouve, par exemple, en synthèse d'image et en vision par ordinateur. Son rôle est de transformer un ensemble de données depuis l'espace 3D vers une image 2D [HZ04a].

Concrètement, étant donné un point $\mathbf{X}$ définit dans l'espace par ses coordonnées $\mathbf{X} = (X, Y, Z)^T$, son projeté $\mathbf{x}$ sur un plan image situé à une distance f du centre de projection est donné par les coordonnées $\mathbf{x} = (x, y, f)$. Plus particulièrement, on obtient $\mathbf{x} = (fX/Z, fY/Z, 1.0)$ en restreignant les coordonnées à un repère associé au plan image centré sur le point principal[13] p comme décrit sur la figure 59. La projection peut alors être décrite sous la forme d'une relation matricielle en coordonnées homogènes $\mathbf{x} = P\mathbf{X}$ tel que :

[13]Point à l'intersection du plan image et de l'axe principal.

$$\begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.1}$$

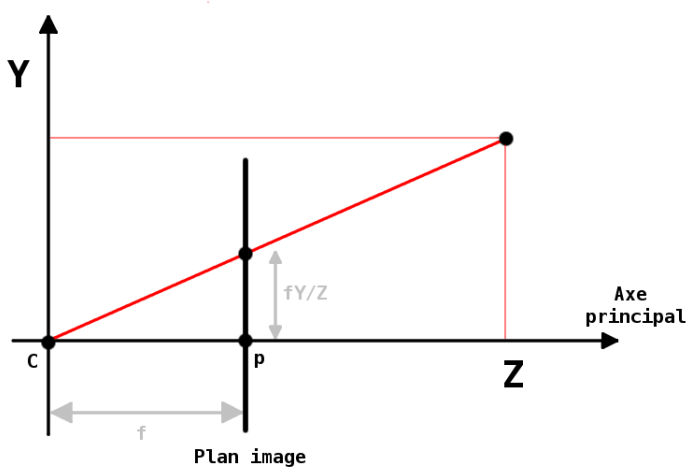


FIG. 59: Projection d'un point sur le plan image

Habituellement le repère associé au plan image n'est pas centré sur le point principal mais dans le coin inférieur gauche. Le changement de référentiel s'opère en désignant la projection comme :

$$\begin{pmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{pmatrix} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.2}$$

Cette relation peut aussi s'écrire de la manière suivante :

$$\mathbf{x} = \mathtt{K}[\mathtt{I}|0]\mathbf{X} \text{ avec } \mathtt{K} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

La matrice K obtenue juste au-dessus est connue sous le nom de «matrice des paramètres intrinsèques». Sur certains types de caméras les capteurs CCD n'ont pas forcément les axes x et y exactement perpendiculaires. Un paramètre s, pour *skew* (facteur de verticalité), est alors introduit dans la matrice K pour tenir compte de cette déformation. Définir s comme nul signifie que les deux axes sont perpendiculaires, ce qui est le cas avec la plupart des capteurs. On a alors :

$$\mathtt{K} = \begin{bmatrix} f & s & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

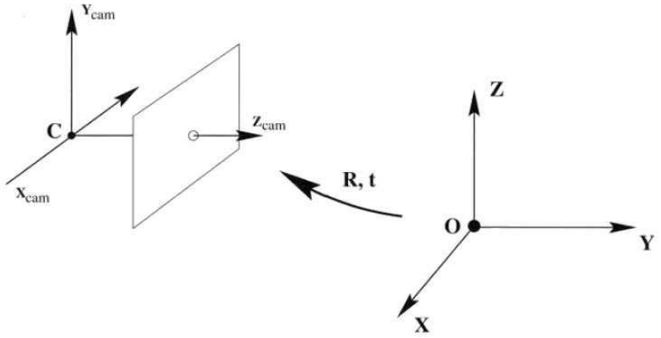


FIG. 60: Les paramètres extrinsèques permettant d'effectuer le changement de référentiel *Copyright : Hartley et Zisserman*

Généralement un point de l'espace possède des coordonnées dans un repère donné différent de celui de la caméra. Notre relation de projection précédente étant donnée dans le repère de la caméra, il est nécessaire d'effectuer un changement de référentiel qui se constitue d'une rotation et d'une translation qui sont connues sous le nom de «paramètres extrinsèques» (figure 60). La rotation $\mathtt{R}$ est définie par une matrice 3×3 et la translation par le vecteur $\mathbf{t} = -\mathtt{R}\mathbf{C}$ où C est le centre de la caméra dans le référentiel de la scène. On obtient alors pour le changement de référentiel :

$$\mathbf{X_{cam}} = \begin{bmatrix} \mathtt{R} & -\mathtt{R}\mathbf{C} \\ 0 & 1 \end{bmatrix} \mathbf{X} \tag{2.5}$$

La matrice de projection P va donc se définir comme l'association des paramètres intrinsèques et des paramètres extrinsèques de la caméra tel que :

$$\mathbf{x} = \mathtt{P}\mathbf{X} = \mathtt{KR}[\mathtt{I}|-\mathbf{C}]\mathbf{X} = \mathtt{K}[\mathtt{R}|\mathbf{t}]\mathbf{X} \tag{2.6}$$

2.1.2 Calibrage par la méthode de Zhang

Le calibrage d'une caméra consiste à déterminer les valeurs de ses paramètres extrinsèques et intrinsèques. Ces derniers ne dépendent que du matériel (à condition de ne pas zoomer et sans focus) de chaque caméra ce qui signifie, à la différence des paramètres extrinsèques, que la position et la direction de la caméra n'ont pas d'influence. Plus concrètement, il ne sera nécessaire de calculer ces paramètres qu'une seule fois et ils pourront

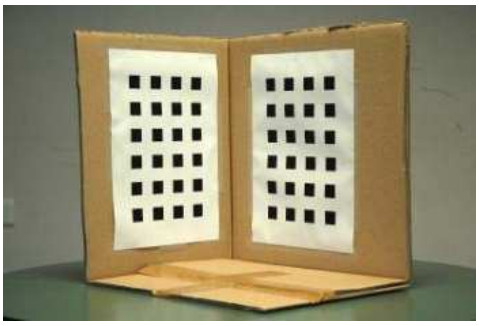


FIG. 61: La mire de calibrage de Tsai *Copyright : Vincent Nozick*

être réutilisés pour chaque nouveau calibrage d'une même caméra.

Pour calibrer une caméra, différentes méthodes ont été proposées comme le *Gold Standard Algorithm* de Hartley et Zisserman [HZ04b] (adaptation de la méthode *Direct Linear Transformation* [AAK71]) ou la méthode de Tsai [Tsa22] qui nécessitent, par exemple, une mire composée de deux damiers positionnés perpendiculairement (figure 61). Dans notre cas, nous avons opté pour la méthode de Zhang [Zha00a] qui est un bon rapport entre précision numérique et facilité d'utilisation. Elle est notamment utilisée dans la bibliothèque *OpenCV* [ope].

Le premier objectif est de calculer la matrice des paramètres intrinsèques. En considérant un ensemble de points coplanaires tel que $Z = 0$ alors l'équation 2.6 peut s'écrire :

$$\mathbf{x} = \mathtt{P}\mathbf{X} = \mathtt{K}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & \mathbf{t}\end{bmatrix}\begin{bmatrix}X\\Y\\0\\1\end{bmatrix} = \mathtt{K}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t}\end{bmatrix}\begin{bmatrix}X\\Y\\1\end{bmatrix} \tag{2.7}$$

Le point $\mathbf{X}$ et son image $\mathbf{x}$ sont liés par une relation homographique $\mathtt{H}$ telle que :

$$\mathbf{x} = \mathtt{H}\mathbf{X} \text{ avec } \mathtt{H} = \mathtt{K}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t}\end{bmatrix} \tag{2.8}$$

En utilisant l'équation 2.8 avec $\mathtt{H} = [\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3]$ et le fait que $\mathbf{r}_1$ et $\mathbf{r}_2$ soient orthonormaux, on peut définir une matrice $\mathtt{B} = \mathtt{K}^{-T}\mathtt{K}^{-1}$ décrivant l'image de la conique absolue telle que :

$$\mathbf{h}_1^T \mathtt{B} \mathbf{h}_2 = 0 \tag{2.9}$$

$$\mathbf{h}_1^T \mathtt{B} \mathbf{h}_1 = \mathbf{h}_2^T \mathtt{B} \mathbf{h}_2 \tag{2.10}$$

$\mathtt{B}$ étant symétrique on peut définir la matrice comme un vecteur $\mathbf{b}$:

$$\mathbf{b} = [B_{11}, B_{12}, B_{22}, B_{13}, B_{23}, B_{33}]^T \tag{2.11}$$

On obtient alors :

$$\mathbf{h}_i^T \mathtt{B} \mathbf{h}_j = \mathbf{v}_{ij}^T \mathbf{b} \tag{2.12}$$

avec $v_{ij} = [h_{i1}h_{j1}, h_{i1}h_{j2} + h_{i2}h_{j1}, h_{i2}h_{j2}, h_{i3}h_{j1} + h_{i1}h_{j3}, h_{i3}h_{j2} + h_{i2}h_{j3}, h_{i3}h_{j3}]^T$.

Pour une homographie donnée et en réutilisant les équations 2.8 et 2.9, on peut écrire :

$$\begin{bmatrix} \mathbf{v}_{12}^T \\ (\mathbf{v}_{11} - \mathbf{v}_{22})^T \end{bmatrix} \mathbf{b} = \mathbf{V}\mathbf{b} = 0 \tag{2.13}$$

où $\mathbf{V}$ est une matrice de taille $2n \times 6$ avec n le nombre d'images de la mire permettant de calculer n homographies. Plus n est grand plus la solution du système sera précise.

En résolvant le système de l'équation 2.12 via l'utilisation de la méthode de décomposition en valeurs singulières (SVD), il est possible d'obtenir une solution du vecteur $\mathbf{b}$, qui est appxroximation au sens des moindres carrés. Sachant que $\mathtt{B} = \mathtt{K}^{-T}\mathtt{K}^{-1}$ et le lien avec le vecteur $\mathbf{b}$, il est alors possible d'en déduire les composantes de la matrice des paramètres intrinsèques $\mathtt{K}$.

La matrice des paramètres extrinsèques peut être déduite en utilisant la relation de l'équation 2.7 avec la matrice $\mathtt{K}$ qui vient d'être calculée. En se fondant sur une homographie donnée, nous allons donc obtenir :

$$\mathbf{r}_1 = \mathtt{K}^{-1}\mathbf{h}_1 \mathbf{r}_2 = \mathtt{K}^{-1}\mathbf{h}_2 \mathbf{t} = \mathtt{K}^{-1}\mathbf{h}_3$$

La matrice $\mathtt{R}$ étant orthonormée on peut en déduire que $\mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2$. Un résultat visuel de calibrage employant la méthode de Zhang est présenté dans la figure 62

2.1.3 Calibrage multi-caméras

Dans le cas du calibrage d'une unique caméra, le référentiel de l'espace 3D est généralement positionné sur un point de la mire utilisée pour le calcul des paramètres extrinsèques. Lorsqu'on utilise un petit nombre de caméras, il est encore possible que chacune d'elles puissent visualiser les points d'une même mire mais sous des angles différents. Dans ce cas chaque caméra sera calibrée dans le même référentiel.

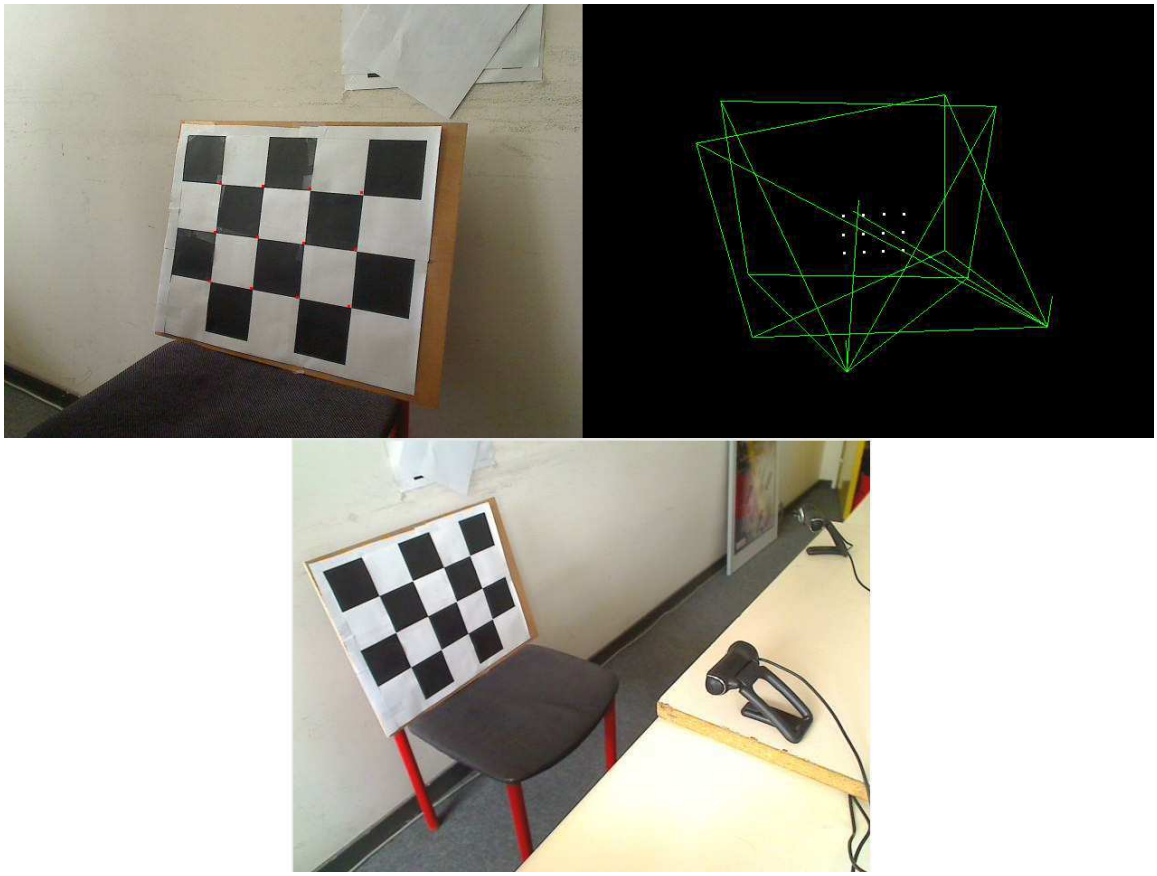


FIG. 62: Exemple de résultat du calibrage de deux caméras utilisant la méthode de Zhang.

Pour une reconstruction utilisant la méthode des *visual hulls*, les caméras sont généralement positionnées en différents points d'une sphère virtuelle avec le sujet en son centre. Il devient logiquement impossible de calibrer les caméras en utilisant une mire qui puisse être visible par chacune d'elles en même temps. Pour outrepasser cette limite nous proposons une solution permettant de calibrer les caméras qui ne peuvent voir la mire de référence en se basant sur celles définies dans ce référentiel.

Soit C_R une caméra, dite de référence, initialement calibrée dans un repère connu R_1 et C_C une caméra calibrée dans un repère R_2. Le principe est de calculer la matrice de passage du repère R_2 au repère de référence R_1 en se basant sur les paramètres de C_R. Si nous arrivons à calibrer cette dernière dans le même référentiel R_2 que C_C, alors il est possible d'appliquer la même matrice de passage à C_C pour qu'elle soit calibrée dans le référentiel R_1.

Concrètement la méthode peut être décrite de la manière suivante :

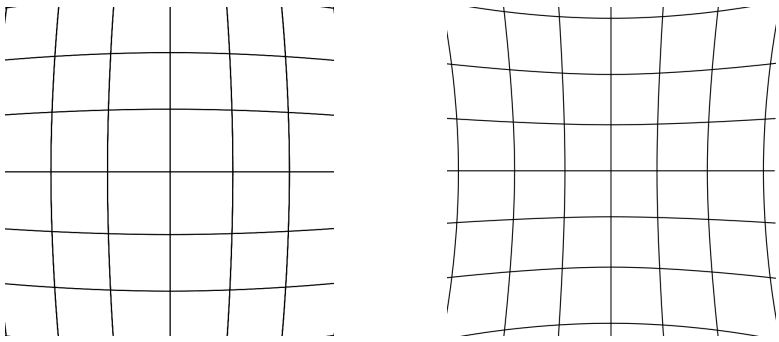
Algorithm 2.1.1: CALIBRAGE MULTI-CAMÉRAS(Cc, Cr)

$E_1 \leftarrow$ paramètres extrinsèques de C_R dans R_1
$E_2 \leftarrow$ paramètres extrinsèques de C_R dans R_2
$M \leftarrow E_2^{-1} \times E_1$
$E_3 \leftarrow$ paramètres extrinsèques de C_C dans R_2
comment: E les paramétres extrinsèques de C_C dans R_1
$E \leftarrow E_3 \times M$

On obtient alors deux caméras définies dans un référentiel connu. Pour calibrer les autres caméras dans ce même référentiel, il suffit d'appliquer cette méthode en se servant d'une des deux caméras. Ainsi toutes les caméras peuvent être calibrées dans un référentiel commun, même si elles ne peuvent pas voir la mire qui sert à le définir.

2.1.4 Correction de la distortion radiale

La plupart des systèmes optiques subissent des deformations en raison de la présence d'une lentille. On peut alors constater que, dans les images obtenues, les lignes supposées droites ont l'apparence de courbes. Il est possible de distinguer deux sortes de distortions radiales qui sont celles dites en «barillet» ou celles dites en «coussin» comme illustré sur la figure 63.



(a) (b)

FIG. 63: Exemples de deformation de l'image provoquées par un système optique : (a) présente une distortion en «barillet» et (b) une distortion en «coussin»

Lorsque cette distortion est trop importante, il est nécessaire de la corriger pour éviter de futures erreurs de calculs comme dans le calibrage de caméras. Un ensemble de solutions est proposé dans [HZ04b].

2.2 La méthode *plane sweep*

Cette méthode fût introduite par Collins en 1996 [Col96] dans l'optique de reconstruire la géométrie des bâtiments en utilisant les contours binaires extraits depuis différentes prises de vue aériennes (figure reffig :ps1).

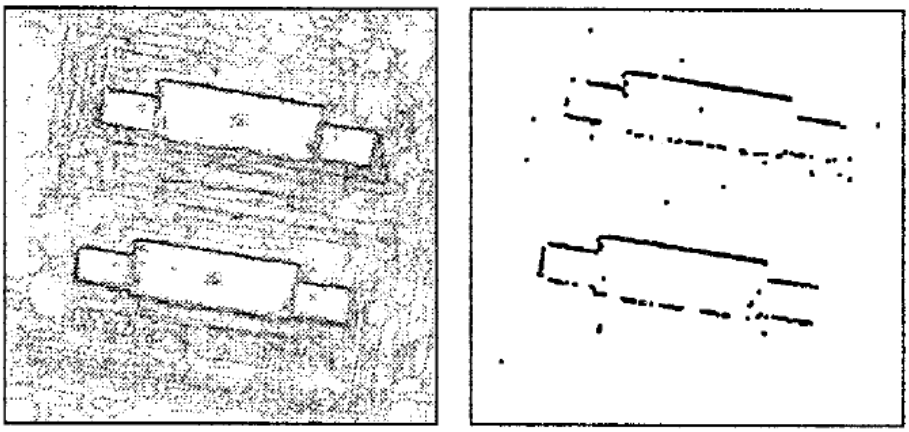


FIG. 64: Un des premiers résultats par la méthode des *planes sweep* associé à un plan précis *Copyright : Collins*

Le principe des *planes sweep*, ou balayage par plans, est de pouvoir générer un ou plusieurs points de vue d'une scène à partir d'un ensemble de caméras. Considérant une scène uniquement composée d'objets diffus, l'utilisateur doit «positionner» la caméra virtuelle Cam_{vir} au milieu des caméras réelles, ainsi que définir les plans *near* et *far* délimitant la scène. L'espace situé entre ces deux plans est ensuite divisé en plans parallèles P_i comme illustré sur la figure 65. Si la surface d'un objet de la scène est situé au niveau d'un plan en un point M, il aura probablement la même couleur depuis n'importe quelle caméra. Dans le cas contraire, si M n'appartient plus à la surface de l'objet mais qu'il est situé sur un plan, alors sa couleur sera sûrement différente d'une caméra à l'autre.

L'hypothèse de l'algorithme de *plane sweep* est alors la suivante : Si la projection sur l'ensemble des caméras d'un point, appartenant à un plan P_i, permet d'obtenir toujours la même couleur alors nous pouvons, a priori, en déduire qu'il correspond à la surface de l'objet.

Pendant le processus de création de la nouvelle vue, chaque plan P_i est évalué depuis le plus éloigné vers le plus proche. Tout d'abord, chaque pixel de P_i est projeté sur les images des caméras. Puis, un score et une couleur sont calculés en tenant compte des correspondances calculées entre les couleurs obtenues. Ensuite, les résultats sont projetés sur la caméra virtuelle Cam_{vir} en se fondant sur le principe du test de profondeur. L'évaluation du plan P_{i+1} intervient ensuite. Finalement, le résultat est prêt une fois que l'ensemble des plans ont été traités.

L'algorithme peut alors être écrit de la manière suivante :

Algorithm 2.2.1: MÉTHODE DE PLANE SWEEP(Cam_{vir}, P, Ref)

comment: Cam_{vir} la position de la caméra virtuelle

comment: P la liste des plans parallèles au plan image de Cam

comment: Ref les images de référence

for each $i \in P$

do { **for each** $pixel \in P_i$

do { projeter $pixel$ sur Ref ; $s_(pixel, i) \leftarrow score(pixel)$ } **for each** $pixel \in Cam_{vir}$

do $color(pixel) \leftarrow max(s_(pixel, i))$

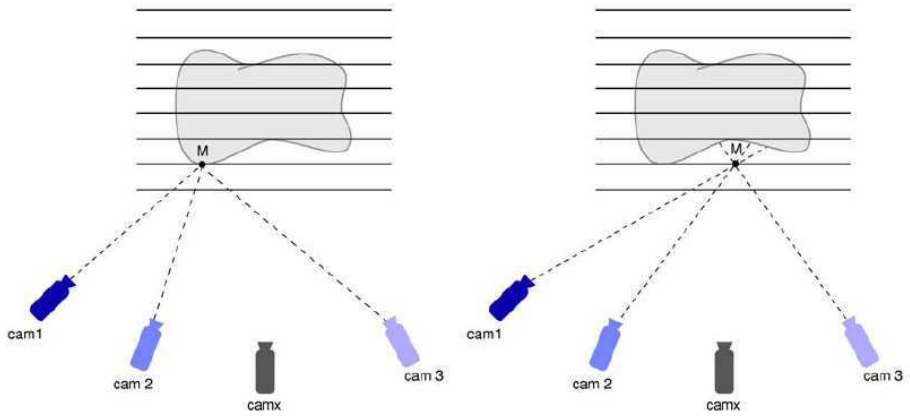


FIG. 65: Sur l'image de gauche le point M est présent sur un des plans et sur la surface d'un objet et contrairement à l'image de droite, peut être utilisé comme résultat sur le point de vue que l'on cherche à obtenir *Copyright : Nozick*

Pour le calcul du *score*, différentes méthodes ont été proposées. Yang *et al.* [YWB02] proposent d'utiliser les *register combiners*[14] afin d'accélérer le rendu. Tout d'abord, une image de référence, définie comme étant celle associée à la caméra dont la position est la plus proche du point de vue virtuel, est déduite. Pour chaque plan D, les images des autres caméras sont alors projetées et un *score* est ensuite calculé pour chacun des points de D comme étant la somme des différences au carré[15] entre ces dernières et l'image de référence puis stocké dans la valeur *alpha*. La moyenne des couleurs des différentes images est également conservée pour chaque point de D. À chacun des pixels de l'image, associée au point de vue que l'on cherche à obtenir, sera attribué le point dont le *score* est le meilleur (figure 66). L'avantage de la méthode de Yang *et al.* est qu'elle fait pleinement

[14] Ancienne version des shaders

[15] ou *SSD* : *Sum of Squared Difference*

usage de la carte graphique car, en plus de calculer le *score*, ils l'utilisent pour effectuer les projections des images des différentes caméras à l'aide d'une homographie.

FIG. 66: Nouveau point de vue d'une scène obtenu à l'aide de l'amélioration apportée sur le principe des *planes sweep Copyright : Yang*

Woetzel *et al.* [WK04] utilisent le principe des *planes sweep* pour générer une carte de profondeur de la scène. Contrairement à la méthode précédente, il n'y a pas d'utilisation d'une vue de référence et le calcul de la somme des différences au carré est appliqué à l'ensemble des paires de vues. Parmi toutes les *SSD* calculées, seules celles ayant un bon score sont conservées ainsi que leur contribution colorimetrique au résultat final. Cette optimisation va permettre de commencer à gérer les occlusions. Une nouvelle optimisation de la méthode de Woetzel *et al.* consiste à ne plus stocker les résultats des *SSD* dans le canal alpha mais plutôt comme la valeur de profondeur. Cette modification va permettre de comparer les *scores* entre chaque plan de manière automatique à l'aide de l'utilisation du test de profondeur. Un exemple de résultat de cette méthode est présenté dans la figure 67.

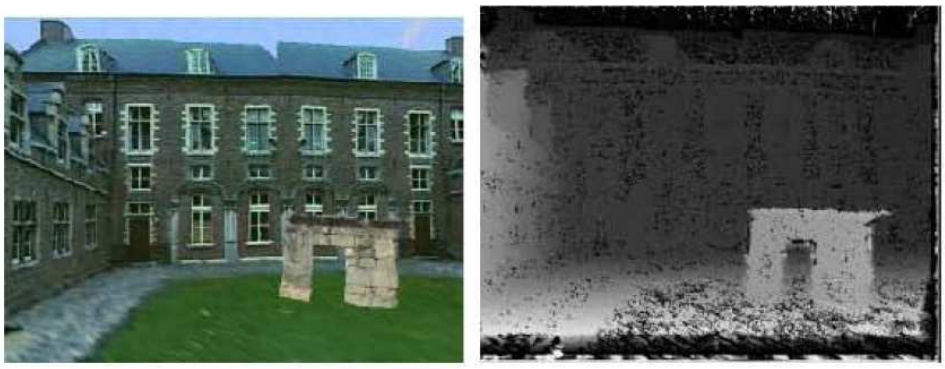


FIG. 67: Création d'une carte de profondeur en utilisant le principe des *planes sweep Copyright : Woetzel*

La méthode de Geys *et al.* propose d'utiliser le principe des *planes sweep* de manière à répartir les charges de calcul entre le *GPU* et le *CPU* [GKG04]. Pour cela, ils proposent

de créer une nouvelle vue en utilisant deux caméras et un déroulement de l'algorithme qui suit deux étapes. La première fait appel à la méthode des *planes sweep* et au *GPU* pour calculer une carte de profondeur à l'aide d'une somme des différences absolues[16], la seconde consiste à appliquer une fonction de minimisation d'énergie sur *CPU* afin d'améliorer la carte de profondeur se fondant sur la continuité spatiale et temporelle, des scores précédemment calculés et d'un terme d'occlusion déduit de la répartition des éléments de la scène entre le premier et l'arrière plan. Finalement un plaquage de texture est effectué sur la carte de profondeur afin de créer la nouvelle vue (figure 68). Il est à noter qu'une version de cette méthode utilisant trois caméras a été proposée dans [GVG04].

FIG. 68: Génération d'une nouvelle vue en se fondant sur une carte de profondeur générée par la méthode des *planes sweep Copyright : Geys*

Nozick *et al.* [NMA06] améliorent les méthodes précédentes en ne calculant plus les *scores* par paire mais sur l'ensemble des images de référence. Trois propositions sont faites pour évaluer le *score* sur chacun des pixels de chaque plan. La première considère le *score* comme étant la variance des couleurs des vues des projections des différentes caméras, la seconde consiste à appliquer une technique de *sigma clipping*. L'algorithme débute par le calcul de la variance sur l'ensemble des couleurs S puis en déduit un *score*. Le *score* le plus éloigné de la moyenne des couleurs de S est ensuite sélectionné pour être supprimé si sa distance est supérieure à une constante prédéfinie. Cette partie est itérée tant qu'un élément de S satisfait les conditions et tant que le nombre d'éléments de S est supérieur à 2. La dernière méthode des auteurs commence de la même manière que la proposition précédente, mais cette fois chaque élément de l'ensemble S est évalué pour savoir si sa contribution améliore le *score*. Si ce n'est pas le cas, il est supprimé du résultat. L'itération est appliquée tant que le nombre d'éléments de S est supérieur à 2 et tant qu'il existe un élément de S pouvant être supprimé.

Nozick *et al.* proposent également d'appliquer leur méthode à la génération d'images stéréoscopiques en considérant que les calculs des *scores* et des couleurs sur les différents plans sont les mêmes pour deux caméras virtuelles à la condition que leur axes principaux

[16] *SAD : Sum of Absolute Differences*

soient perpendiculaires à ces mêmes plans. La seule différence va être au niveau de la projection des plans sur les deux caméras.

FIG. 69: Exemple de 10 vues générées à partir de 4 images *Copyright : Nozick*

2.3 Reconstruction par lumière structurée

Une lumière structurée peut être qualifiée comme étant un patron lumineux tel un ensemble de points, de lignes ou une grille projetée par l'intermédiaire d'un LASER ou d'un video-projecteur. Connaissant la position du projecteur et celle du dispositif de capture (une caméra par exemple), il va être possible d'effectuer une reconstruction en se fondant sur les déformations subies par une lumière structurée envoyée sur un objet [VO90, Kap03]. En conséquence, la scène dans laquelle se trouve l'objet doit être plongée dans le noir et uniquement éclairée par le patron lumineux. L'emploi d'un video-projecteur [RCM+01] pour projeter ce dernier contraint à une utilisation dans un espace réduit à cause de la faible profondeur de champ de ce type d'instrument. Au contraire, le LASER n'a pas ce type de contrainte mais nécessite une bonne synchronisation entre l'émetteur et la caméra. La raison est qu'un LASER a besoin d'un temps minimal pour pouvoir afficher un patron lumineux dans son intégralité, ce qui implique que la vitesse de capture de la caméra doit être adaptée en conséquence.

Dans la plupart des systèmes optiques, il existe des déformations provoquées par les lentilles que l'on nomme distorsions radiales. Les conséquences sont que la projection d'une grille, initialement constituée de carré, va subir un ensemble de déformations qui peuvent fausser les résultats. La solution consiste donc à appliquer un calibrage sur les différents matériels. Dans le cas d'un projecteur il faut déformer de manière uniforme l'image de façon à obtenir une projection correcte.

Après une phase de triangulation, qui consiste à détecter les correspondances entre l'image capturée avec la caméra et le patron projeté [PvGO96b] et à déterminer la position des points caractéristiques de la grille projetée dans l'espace, le modèle est reconstruit. Deux méthodes peuvent s'appliquer. La première permet une reconstruction en se fondant uniquement sur le nuage de points comme proposé par [HDD+92] ou [BBX95], La seconde

va effectuer une reconstruction en préservant la structure des données acquises comme le suggèrent [Cur98], [SL92], [TL94] ou [RL01]. Un exemple de résultats obtenus par cette méthode est illustré dans la figure 70.

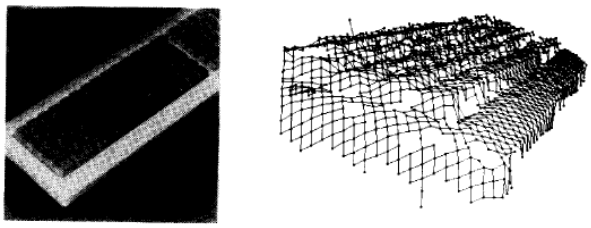


FIG. 70: Exemples de reconstruction d'un clavier à l'aide d'un patron de lumière *Copyright : Vuylsteke*

Il existe une solution proposée par Rusinkiewiz *et al.* qui permet d'effectuer la reconstruction d'un objet quelconque en temps réel. La méthode se fonde sur la projection de plusieurs patrons lumineux à une cadence de 60Hz et une capture faite à l'aide d'une caméra synchronisée. La corrélation des informations issues des différents patrons est assurée par l'utilisation d'un algorithme similaire à celui de l'*ICP* (*Iterative Closest Points*) permettant de conserver des performances élevées. Finalement la phase de rendu est effectuée par le biais de *splat* afin d'éviter d'avoir à calculer les relations entre les différentes composantes du nuage de points.

La principale limite des méthodes fondées sur les patrons lumineux est la difficulté à reconstruire un modèle en prenant en compte les informations chromatiques. Cela est notamment illustré par la nécessité d'utiliser ce type de système soit dans l'obscurité, soit avec des objets généralement blancs. Les solutions habituellement proposées effectuent la capture en deux étapes en commençant par la création de la géométrie puis en terminant par une phase d'éclairage durant laquelle une texture peut être produite. Toutefois, cette démarche ne permet pas d'effectuer une capture en temps réel et encore moins sur des éléments potentiellement mobiles.

Il est à noter qu'une variante des méthodes utilisant les lumières structurées a été proposées par Schindler [Sch08] et reprend les travaux de Woodham [Woo89]. Nommée «stéréoscopie photométrique», le principe est d'éclairer un objet de différentes manières afin de reconstruire les normales à sa surface à partir desquelles il est possible de récupérer la géométrie. Dans le cas présent, l'illumination est provoquée par l'intermédiaire de l'écran d'un ordinateur portable tandis que la capture est assurée par une webcam. La reconstruction est fondée sur la création de la carte de profondeur en appliquant la méthode de relaxation de Gauss-Seidel. Un résultat obtenu via cette méthode est présenté dans la figure 71. Pour atteindre des résultats en temps interactif, l'auteur soumet l'idée d'utiliser une illumination multi-spectrale pour obtenir trois variations d'illumination en une seule passe.

FIG. 71: Exemple de reconstruction utilisant la méthode de stéréoscopie photométrique *Copyright : Schindler*

2.4 Animation squelettique

Une dernière catégorie de méthode destinée à créer un avatar depuis un sujet réel se nomme le «*motion capture*». Bien connu dans le domaine cinématographique depuis quelques années et plus récemment celui du jeu vidéo, le *motion capture* se fonde sur la détection et le suivi de points caractéristiques associés à un individu. Ces points sont ensuite associés entre eux de manière automatique [MBT96] pour former un squelette qui est utilisé dans l'animation d'un personnage virtuel. Traditionnellement, le sujet principal du *tracking* est obligé de revêtir une tenue particulière qui est composée de multiples boules reflétant par exemple une lumière infrarouge [KOF05] (figure 72) ou tout autre capteur. La transposition de ces éléments du réel vers le virtuel est rendue possible grâce aux nombreuses caméras calibrées réparties autour de l'espace dans lequel le sujet évolue.

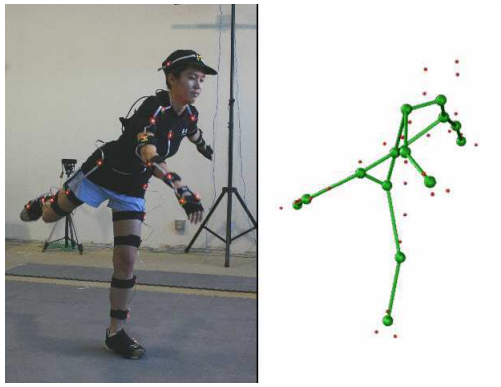


FIG. 72: Exemple de *motion capture* utilisant des marqueurs optiques pour générer un squelette *Copyright : Kirt et al.*

Dans le cadre de la réalité virtuelle, le port d'une combinaison est à éviter tout d'abord parce qu'elle ne facilite pas à mettre l'utilisateur en condition pour qu'il puisse réagir naturellement, ensuite parce que la tenue est en elle-même relativement contraignante à porter. Pour que le *motion capture* puisse être correctement employé en réalité virtuelle il faut que la méthode puisse s'appliquer sans qu'elle oblige l'utilisateur à porter quoique ce soit durant l'expérience. Pour cela différentes techniques de capture de mouvements sans marqueur ont vu le jour, dont un aperçu peut être obtenu dans [MG01] et [MHK06].

Parmi les solutions en temps réel, nous pouvons citer les travaux de Ballan et Cortelazzo [BC08]. L'algorithme proposé repose sur la création d'un «arbre kinématique» qui permet d'organiser hiérarchiquement les os du squelette entre eux. En servant des informations 2D tirées des différentes silhouettes obtenues depuis les caméras positionnées autour de l'espace de travail et du flux optique correspondant aux variations de position en fonction des captures précédentes à $t-1$. Pour chacun des points de vue, une correspondance est calculée entre les sommets du modèle géométrique et les silhouettes afin de récupérer une matrice de transformation de l'espace 3D vers l'espace 2D. L'ensemble des informations obtenues sont recoupées dans une fonction dont le résultat, minimisé par l'utilisation de la méthode de Levenberg-Marquardt, permet d'obtenir une configuration correcte du squelette comme sur la figure 73. Le modèle de l'utilisateur est finalement transformé en conséquence pour correspondre à la posture du squelette.

FIG. 73: Exemple de *motion capture* n'utilisant pas de marqueur *Copyright : ballan et cortelazzo*

Pour garantir une cohérence visuelle entre l'utilisateur et son modèle, la phase de *motion capture* est généralement précédée d'une étape de reconstruction unique qui passe par l'utilisation d'un scanner [BM02]. Le sujet est placé au sein d'un système qui à l'aide d'un LASER va reconstruire une version virtuelle de l'individu comme décrit dans [WW03b] et [JWS00]. Un autre procédé se fonde sur l'emploi d'une caméra prenant des vues de l'utilisateur et en les associant par exemple à une lumière structurée comme le décrit [BDD+00]. Ce passage nécessaire à l'obtention d'une copie de l'utilisateur n'est pas effectué en temps réel pour permettre une reconstruction plus fine qui sera animée grâce au squelette calculé. Toutefois, le résultat ne tiendra pas compte des diverses variations comme les mouvements présents au niveau du visage, de bouche ou des yeux.

2.5 Les *Visual hulls*

D'une manière générale un *visual hull*, introduit pour la première fois par Laurentini [Lau91], pourrait être décrit comme le résultat de la corrélation d'informations issues de plusieurs caméras. La silhouette d'un sujet filmé dans un environnement connu est tout d'abord extraite à l'aide d'une méthode de soustraction d'arrière plan sur chacun des flux vidéo. Pour chaque silhouette, un volume ayant l'aspect d'un cône est créé sur la base des limites de la silhouette et dont le sommet est centré sur la position de la caméra associée, comme illustré sur la figure 74. De cette manière l'espace est réduit à un volume dans lequel le sujet se trouve. En calculant l'intersection de tous ces volumes, on obtient une approximation de la forme recherchée mais excluant les zones concaves. Ainsi plus il y aura de caméras réparties autour du sujet, plus l'approximation de la forme 3D sera précise.

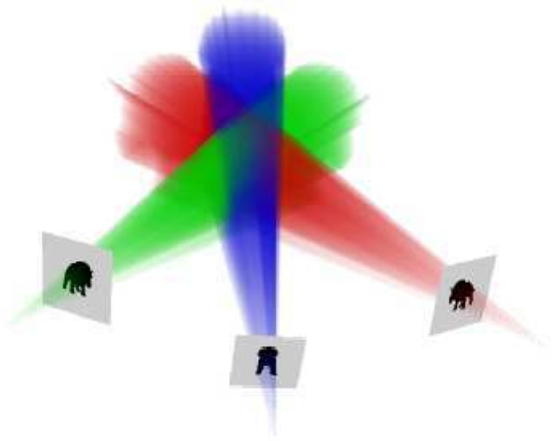


FIG. 74: Illustration de la reconstruction par la méthode des *visual hulls Copyright : Matusik*

Formellement, la création d'un *visual hull* (VH) peut être réduit à $VH(\mathsf{I}, \mathsf{C}) = \bigcap_{i \in \mathsf{I}, j \in \mathsf{C}} V_j^i$ avec I l'ensemble des images, C l'ensemble des contours et V le cône de déprojection. Une définition plus rigoureuse tenant compte de multiples cas particuliers est donnée dans [FB03]. L'ensemble des différentes méthodes existantes pour la réalisation d'un *visual hull* sont classifiables en trois catégories que nous allons présenter.

2.5.1 *Volumetric visual hulls*

La première catégorie de méthodes est appelée *volumetric visual hulls* ou *space carving* et s'appuie sur un découpage de l'espace sous forme de voxels[17]. Seuls les voxels présents à l'intersection des volumes créés à partir des silhouettes sont conservés. L'ensemble ainsi

[17] Terme désignant le plus petit élément discernable d'un espace en trois dimensions.

obtenu forme la représentation 3D de l'objet ou de l'individu filmé. Parmi les travaux initiateurs de cette méthode, on peut citer ceux de Potmesil [Pot87], de Kanade *et al* [KSV98] et ceux de Borovikov, Davis [BD00] et Dyer [Dye01]. Un survol des méthodes existantes peut également être vu dans [Eis06].

Une caractéristique majeure de cette méthode est la difficulté à produire et conserver le rendu d'un résultat en temps réel. À l'image de l'installation mise en avant par la méthode de Theobalt *et al.* [TLMS03], l'utilisation d'un ordinateur unique pour récupérer les flux optiques et effectuer la reconstruction est très occasionnelle. Une architecture reposant sur de multiples PCs permet de répartir les différentes opérations de la chaîne de traitements. Une configuration possible est d'associer deux caméras à une unité de calculs qui va délimiter les silhouettes et produire une première reconstruction grossière sur cette base. Les différents résultats sont ensuite transmis via le réseau à un PCs qui va centraliser les pré-modèles pour finalement en produire un plus fin. Il devient possible avec ce principe d'ajouter de nouvelles caméras tout en conservant la rapidité de rendu du résultat.

Le meilleur moyen pour obtenir une reconstruction en temps réel ou en temps interactif consiste à employer un grand nombre d'unités de calcul qui vont se répartir les différentes parties du processus de capture et de rendu. L'amélioration des cartes graphiques a permis d'accroître la puissance de calcul de chacun des PCs mais à la condition de pouvoir adapter l'algorithme à l'architecture des *GPU*. Dans le cas des *volumetric visuall hulls*, Zach et Karner [ZKRB05] proposent d'utiliser les *shaders* sur certaines parties de leur méthode. La reconstruction est assurée par une division de l'espace en plans qui vont être parcourus de l'avant vers l'arrière. À chaque traitement d'un des plans, un calque de voxels est calculé en se basant sur la consistance des pixels projetés depuis l'ensemble des vues et d'une carte de profondeur associée à chaque vue qui se complète au fur et à mesure des parcours des plans. La mise à jour de ces dernières est effectuée par un programme sur les fragments qui va rendre ou non transparent un voxel en fonction de son appartenance au modèle qui est en reconstruction. Nitschke *et al.* [NNT07] accèlerent les calculs de la méthode de *space carving* en utilisant les *Vertex Buffer Objects*[18], les *FBO* et un programme sur les fragments pour évaluer si un voxel doit être considéré comme transparent ou opaque. Dans ce dernier cas, la couleur du voxel est également déduite à l'aide d'un *shader*, mais nécessite de stocker l'ensemble des images des caméras dans des textures. Malgré cela, les résultats peuvent être doublés par rapport à ceux obtenus via l'application du même algorithme sur CPU.

L'utilisation d'un maillage composé de triangles est parfois plus avantageuse que de conserver un ensemble de voxels. C'est notamment le cas lorsqu'il s'agit de produire un modèle sur lequel on désire ajouter une texture, appliquer un modèle d'illumination ou lorsqu'il y a besoin de le faire interagir avec son environnement. L'utilisation de triangles se révèle dans ce cas plus adaptée. Hasenfratz *et al.* [HLGB03] ont donc proposé une méthode de reconstruction visant cet objectif mais restant fondée sur une première étape

[18]Méthode permettant de décrire efficacement les primitives géométriques et d'accélérer leur rendu.

utilisant les voxels. Le second but de leur méthode est de recréer une approximation de la forme 3D en temps réel. Après l'étape d'extraction de l'arrière plan pour isoler les silhouettes, la voxelisation s'effectue en utilisant une technique de plaquage de textures, inspirée des travaux de Lok [Lok01], sur une série de plans discrétisant un espace délimité (figure 75). L'application de l'opération logique *ET* sur le résultat de la projection va produire un ensemble de points dans l'espace dont chacun représente un voxel. Finalement un maillage constitué de triangles est produit en utilisant l'algorithme des *marching cubes*[19] sur les voxels. Pour une décomposition de l'espace en 64^3 voxels, ils obtiennent un résultat avoisinant les 30 images par seconde en utilisant une SGI Onyx3000-IR3 pour la reconstruction ainsi que 4 PCs reliés chacun à une caméra.

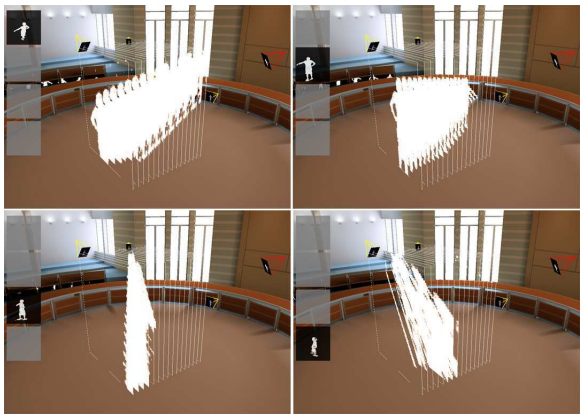


FIG. 75: Recontruction en utilisant le plaquage de textures sur des plans discrétisant un espace donné *Copyright : Hasenfratz et al*

Certaines méthodes ont toutefois été développées afin de pouvoir ajouter de la couleur sur le résultat de la reconstruction sans avoir à recourir à l'utilisation de triangles. Wurmlin et al [WLSG02] proposent par exemple d'utiliser un rendu par points (*splatting*) en utilisant les informations stockées dans un *octree*[20] et créé à partir des différents flux vidéo. Yamazaki *et al* [YSK+02] proposent, quant à eux, d'utiliser des facettes pour approximer un modèle et font en sorte de toujours les orienter vers l'observateur (*billboarding*). La couleur de chaque facette est déduite depuis les informations extraites des caméras et du point de vue désiré. S'inspirant de cela, Goldlücke et Magnor [GM03] ont amélioré la méthode en ajoutant de la transparence aux facettes situées sur les bordures des silhouettes, en l'appliquant au contexte des *volumetric visual hulls* en remplaçant chaque voxel par une facette. Un exemple d'utilisation de cette méthode est présenté dans la figure 76.

[19] Algorithme permettant de générer un objet polygonal à partir d'un champ scalaire en trois dimensions.

[20] Arbre dont chaque nœud possède soit 8 fils, soit aucun.

FIG. 76: Recontruction obtenue par la méthode des micro-facettes *Copyright : Goldlücke et Magnor*

2.5.2 *Polyhedral visual hulls*

Nous avons vu dans la section précédente qu'il était possible d'obtenir un maillage en utilisant une méthode initialement destinée à produire un résultat constitué de voxels. Cette solution n'est toutefois pas la plus efficace pour arriver à cette fin, car elle requière une installation composée de nombreux ordinateurs pouvant répartir les charges de calculs et ainsi obtenir un affichage en temps réel. Afin d'obtenir une solution qui soit uniquement composée de facettes, diverses méthodes ont été proposées.

D'une manière générale, la création d'un *visual hulls* constitué de triangles, se fait grâce à l'intersection des différents volumes coniques délimités par les silhouettes extraites depuis les différentes caméras. La technique la plus couramment utilisée consiste à appliquer un algorithme du type CSG[21] [GHF86, Wie96, SLJ98]. C'est sous cette forme que Baumgart présenta pour la première fois la méthode dans [Bau75] et réutilisée plus tard par Li *et al.* [LMS03].

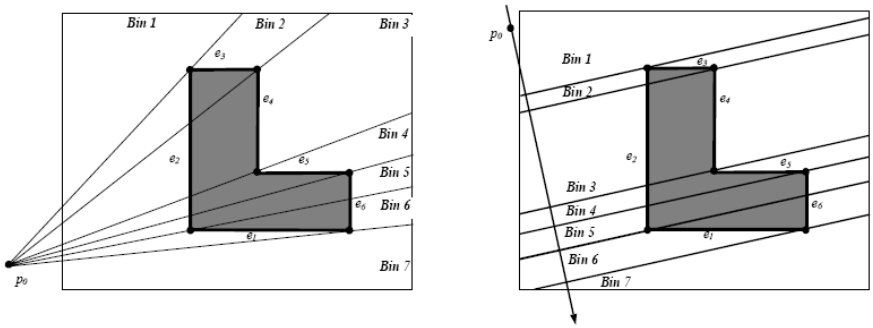
La méthode la plus simple pour déterminer la couleur des facettes, consiste à utiliser une technique de projection de texture [DYB98]. Pour chaque facette, on va projeter dessus le contenu de la silhouette qui est le plus proche de la direction de sa normale. Cependant, on comprendra que des incohérences vont apparaître dans le cas d'occlusions. En effet, dans le cas où un objet qui est visible par une caméra et derrière lui se trouve un autre objet, la texture va alors se projeter de la même manière sur les deux objets sans tenir compte des occlusions. Nous verrons un peu plus loin que plusieurs solutions ont été proposées.

Il est possible d'effectuer le calcul des intersections entre les cônes dans un espace 2D afin d'accélérer les calculs. Le principe de base consiste à prendre chaque face d'un des cônes et de le projeter sur les autres vues pour calculer l'intersection entre la silhouette

[21] Constructive Solid Geometry

et la face. De cette manière, cette dernière aura la taille minimale. En appliquant cet algorithme pour chaque face des cônes de chaque vue, on obtient finalement une reconstruction approximative d'un objet ou d'une personne.

Matusik, Buehler et McMillan [MBM01] proposent d'améliorer cette intersection 2D en utilisant la relation épipôlaire [HZ04c] qui existe entre les vues. Pour deux points de vue donnés A et B, il est possible de déterminer un point précis correspondant à la projection de la caméra de A sur B et appelé épipôle. L'espace est ensuite partitionné sous formes de zones delimitées par des lignes issues de l'épipôle et passant par chacun des sommets de la silhouette. De plus à chacune des lignes est associé un angle se basant entre ces dernières et une ligne de référence donnée. Une structure particulière nommée *edge-bin* est utilisée pour stocker les arêtes de la silhouette présentes dans chacune des zones comme illustré dans la figure 77.



Bin	1	2	3	4	5	6	7
Edges	$\emptyset$	e_2, e_3	e_2, e_4	e_2, e_5	e_2, e_6	e_1, e_6	$\emptyset$

FIG. 77: Exemple d'utilisation de la structure *edge-bin Copyright : Matusik, Buehler et McMillan*

En plus de cela, la structure conserve les angles des deux lignes délimitant la zone. L'étape suivante consiste à projeter une face issue du cône A et d'en calculer l'intersection avec la silhouette de B. La face va être définie par deux lignes dont l'origine est l'épipôle. Pour la première ligne on va commencer par chercher à quel *edge-bin* elle appartient et en déduire une première intersection. On parcourt chaque zone et on ajoute chaque sommet présent à l'intérieur au résultat de l'intersection. On applique ce principe jusqu'à ce que la deuxième ligne de la face soit trouvée dans une zone. Un exemple de résultat d'intersection est présenté sur la figure 78. Finalement le résultat de l'intersection est reprojeté sur la face du cône de A.

En appliquant la même opération pour cette face sur chacun des n points de vue, on va alors avoir $n-1$ résultats d'intersection qui vont lui être associés. Pour obtenir le résultat de l'intersection de ces polygones, les auteurs proposent d'utiliser la même décomposition que pour le calcul de l'intersection d'une face avec une silhouette. L'application de

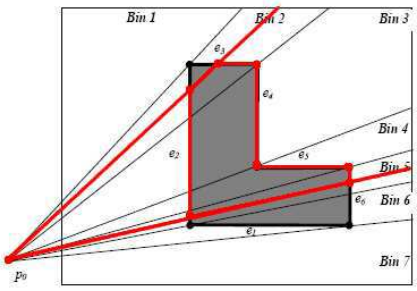


FIG. 78: Résultat de l'intersection d'une face avec une silhouette *Copyright : Matusik, Buehler et McMillan*

ce principe permet de réduire les polygones en de simples quadrilatères (figure 79), ce qui par conséquent diminue la complexité des calculs. L'application de la méthode pour chacune des faces du cône de chaque vue permet d'obtenir le *visual hull*.

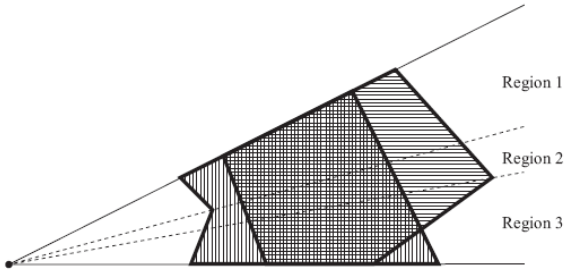


FIG. 79: Intersection de deux polygones constitués de cinq arêtes réduites au calcul de trois intersections de quadrilatères *Copyright : Matusik, Buehler et McMillan*

Afin de garantir un texturage correct du *visual hull*, c'est-à-dire sans les problèmes liés à une simple projection de texture, les auteurs commencent par faire un test permettant de déterminer quelles sont les facettes du polyèdre visibles depuis une caméra donnée. Pour cela, ils profitent du calcul de l'intersection entre une face et une silhouette pour effectuer ce test. Pour chaque *edge-bin*, l'arête la plus proche de l'épipôle est marquée comme visible, les autres sont marquées comme invisibles. L'information est ensuite généralisée à la facette du polyèdre. Le texturage de cette dernière est fait en sélectionnant la caméra étant la plus proche du point de vue depuis lequel le *visual hull* est observé. Une contribution de chacune des caméras est alors calculée en fonction de son écart par rapport à la vue souhaitée. Dans le cas où une des caméras a échoué au test de visibilité pour une facette donnée alors sa contribution sera considérée comme nulle.

Une autre approche de cette méthode a été proposée par Boyer et Franco [BF03] qui réutilisent le principe d'intersection 2D entre les lignes épipolaires et les différentes silhouettes de Matusik *et al.* mais avec la volonté d'obtenir un nuage de points approximant la

FIG. 80: *Visual hull* obtenu par la méthode de [MBM01] *Copyright : Matusik, Buehler et McMillan*

surface de l'objet que l'on cherche à reconstruire. La création de l'enveloppe est obtenue en appliquant la triangulation de Delaunay sur ce même nuage. Dans [FLB06], Franco *et al.* améliorent la détection du nuage de points en cherchant pour chaque droite d issue d'un caméra et passant par un point de la silhouette, les deux droites issues de deux autres caméras pouvant ramener d à un segment de taille minimal tangent à la surface recherchée. Une courbe est construite à partir des trois droites trouvées en se fondant sur l'algorithme [BB97]. Le point de contact avec la surface est alors défini comme étant le point de la courbe où sa pente est identique de chaque côté.

2.5.3 *Image based visual hulls*

La dernière catégorie de visual hulls se classe dans la famille des rendus à base d'images ce qui signifie que l'ensemble des calculs vont se faire dans l'espace de l'image et qu'il n'y aura donc pas de maillage créé ou de partitionnement de l'espace. Matusik *et al.* ont ainsi proposé la méthode [MBR+00] en se fondant sur ce principe.

Pour le calcul du *visual hull*, ils commencent par définir les contours des silhouettes pour chacune des images de référence sous la forme de polylignes. Après avoir défini le point de vue que l'on désire obtenir, un rayon issu de ce dernier et passant par chacun des pixels de l'image désirée va être calculé, comme pour une méthode de lancer de rayons. À l'instar de la méthode [MBM01], les intersections entre les différents rayons et les silhouettes sont trouvées en utilisant la relation épipolaire existante avec les images de référence. Une fois l'intersection 2D calculée, elle est reprojetée en 3D sur le rayon et en ne conservant que les segments produits de l'intersection entre ce résultat et ceux obtenus avec les autres images de référence. Le processus est répété pour chacun des pixels et un ensemble de segments est trouvé et plus particulièrement les extrémités p les plus proches du point de vue désiré sont évaluées.

Pour attribuer une couleur à chacun des pixels, il faut sélectionner une image de référence dont les informations vont pouvoir être prises. Pour chaque pixel de l'image désiré un angle va être calculé entre chacun des points de vue de référence, le point p

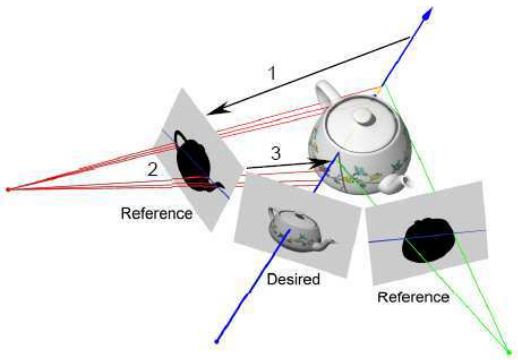


FIG. 81: Calcul des intersections pour chaque rayon issu d'un pixel de l'image désiré *Copyright : Matusik et al.*

courant et le point de vue désiré. L'image de référence, à laquelle l'angle le plus faible est associé, est conservé. Pour résoudre le problème d'occlusion généré par cette méthode, Matusik *et al.* proposent une nouvelle fois de se servir de la relation épipolaire. D'une manière générale, l'ensemble des segments obtenus à l'étape précédente vont être projetés du plus proche au plus éloigné sur une image de référence donnée. Si pour un point p d'un segment, il existe déjà un projeté alors ce point n'est pas visible depuis cette image de référence. Concrètement, pour une image de référence r donnée, les pixels de l'image désirée sont parcourus de droite à gauche ou de gauche à droite afin d'assurer un parcours des segments de l'avant vers l'arrière. Chacun des segments à un pixel est projeté sur l'image de référence et accumulés aux projections déjà effectuées. Dans le cas où il existe déjà un résultat à l'endroit où se projette le point p associé à ces segments alors il est considéré comme caché. Dans le cas contraire il est visible. Cette information est alors prise en compte lors du choix de l'angle pour déterminer la couleur du pixel de l'image désirée.

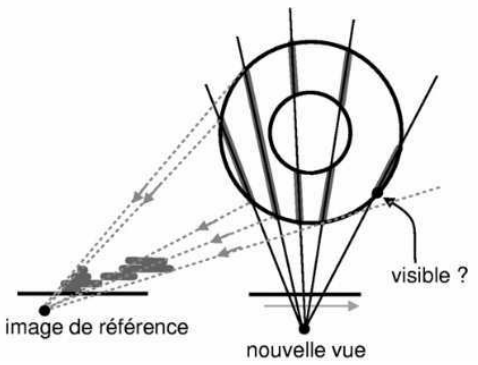


FIG. 82: Evaluation des occlusions à l'aide de la relation épipolaire *Copyright : Matusik et al.*

2.5.4 *Photo hulls*

Il existe une dernière méthode appelée *photo hulls* qui est le résultat du mélange des techniques de *visual hulls* et de vérification de *photo-constitency*. Tout d'abord, introduite par Seitz *et al.* [SD97], la méthode fut améliorée par Li *et al.* [LMS04] en utilisant la carte graphique pour accélérer les temps de calcul. D'une manière générale, la reconstruction ne se fonde plus sur une intersection des volumes créés à partir de silhouettes, mais sur les relations colorimétriques qui existent entre les différentes vues. À l'instar de la méthode des *planes sweep*, une discrétisation de l'espace sous forme de plans faisant face au point de vue recherché est effectuée. Toutefois, cette dernière ne s'applique pas à l'ensemble de l'image mais uniquement sur une boîte englobante associée à l'objet que l'on cherche à reconstruire. Le principal bénéfice de cette méthode est qu'il est possible de générer les parties convexes des objets. Un exemple de résultat faisant appel à cette technique est présenté dans la figure 83.

FIG. 83: Rendu d'une nouvelle vue en utilisant la technique de *photo hulls Copyright : Li et al.*

2.6 Bilan

Nous venons de présenter quatre grandes familles de méthodes qui permettent d'obtenir en temps réel (ou, au pire, en temps interactif) l'avatar d'un individu. Les travaux ont tous en commun de reposer sur l'utilisation de caméras calibrées mais différent sur le principe de reconstruction. Ils ne conviennent donc pas toujours pour les objectifs que nous cherchons à atteindre.

Nous avons commencé par introduire le principe de *plane sweep* [Col96, YWB02, WK04, GKG04, NMA06] qui grâce aux cartes graphiques peut permettre d'obtenir un ré-

sultat en temps réel. La principale contrainte est que la méthode permet d'obtenir un nouveau point de vue d'une scène mais dans son intégralité. Seule le mélange de la technique de *plane sweep* à une autre, comme les *visual hulls* avec les *photo hulls* [SD97, LMS04], permet la «reconstruction» d'un objet isolé. Enfin, le résultat de la méthode de *plane sweep* est obtenu dans l'espace image ce qui est contraignant pour évaluer des interactions avec l'environnement virtuel ou créer des effets comme les ombres.

La reconstruction fondée sur l'utilisation d'une lumière structurée [VO90, Kap03, RCM+01, PvGO96b, HDD+92, BBX95, Cur98, SL92, TL94, RL01] est intéressante dans le sens où elle permet d'obtenir en temps réel un modèle. Toutefois, les contrainte associées sont relativement importantes. Tout d'abord, l'utilisateur est constamment soumis à la projection d'un patron lumineux sur sa personne ce qui peut nuire au sentiment de présence. Ensuite, la méthode nécessite que le sujet soit constamment placé dans l'obscurité pour faciliter la détection des patrons, mais cela implique qu'il est difficile de récupérer des informations chromatiques necessaires au rendu de l'avatar. Enfin, l'utilisation d'une lumière structurée limite l'espace dans lequel l'utilisateur peut evoluer en raison de la taille du patron lumineux.

Nous avons également présenté différentes approches fondées sur l'animation d'un modèle 3D qui est, généralement, acquis lors d'une étape précèdent l'experience de réalité virtuelle [MBT96, KOF05, MG01, MHK06, BC08]. Les mouvements de l'utilisateur sont retranscrit en déduisant une posture de squelette qui est appliqué à l'avatar. La limite de cette famille de méthodes est que le rendu va apparaitre statique car les informations sur les couleurs sont pré-acquises. Ainsi les vetements ou alors les expressions faciales, par exemple, ne vont pas varier au cours du temps, ce qui fait que l'utilisateur peut ne pas s'identifier à l'avatar.

Enfin, nous avons décrit les différents travaux liés à la méthode des *visual hulls*. Nous avons vu qu'il en existait trois variantes qui sont les *volumetric visual hulls*, les *Polyhedral visual hulls* et les *Image based visual hulls*. Parmi celles-ci, nous nous sommes particulièrement intéressé à la deuxième catégorie qui permet de construire un avatar de l'utilisateur en produisant le maillage correspondant en temps réel. De plus, il est possible d'obtenir un rendu fidèle en récupérant les données depuis les différents flux vidéos. Pour ces raisons, notre choix s'est orienté vers l'utilisation de cette méthode pour obtenir un avatar à partir de l'utilisateur.

3

Présence : Notre contribution

Dans ce chapitre, nous allons présenter notre démarche déstinée à améliorer le sentiment de présence de l'utilisateur placé au centre d'un système de réalité virtuelle. Notre objectif est de générer sa copie virtuelle qu'il pourra observer de manière naturelle par le biais de surfaces réflechissantes ou tout autre indices visuels. De cette manière, nous contribuons à stimuler les quatres «piliers» nécessaires à la création du sentiment de présence. Nous proposons d'accroître visuellement l'immersion, en utilisant un avatar interagissant en temps réel avec l'environnement vituel et aux mouvements de l'utilisateur, tout en essayant de succiter l'apparition d'émotions chez ce dernier.

3.1 Création d'un avatar par la méthode des visual hulls

Parmi toutes les techniques de reconstruction fondées sur la méthode des *visual hulls*, celle des *Polyhedral visual hulls* va nous intéresser. Son avantage sur les autres, est de proposer initialement un résultat sous la forme d'un maillage en temps réel. Cela va permettre de pouvoir réutiliser l'avatar sans avoir le besoin de recalculer son enveloppe pour un point d'observation différent. L'emploi d'un *visual hull* sous forme de maillage offre également la possibilité d'avoir à notre disposition une représentation de l'utilisateur pouvant influer sur l'environnement virtuel.

Dans cette partie, nous allons détailler les trois étapes de création d'un *visual hull* ainsi que nos apports. Nous commencerons par étudier la méthode de calibrage des caméras, présenter notre reconstruction et finalement le procédé de rendu.

3.1.1 Estimation des silhouettes

Après avoir effectué le calibrage des différentes caméras, il est possible de passer à la première étape de création d'un *visual hull* qui consiste à extraire des différents flux vidéo, les objets que l'on souhaite reconstruire virtuellement. Pour cela, nous allons utiliser une

technique d'extraction d'arrière plan qui se définit comme un ensemble d'éléments qui ne subit aucune modification au cours du temps.

Les principales contraintes liées à la problématique de l'extraction d'arrière plan sont d'une part, la gestion des variations de luminosité de l'environnement et d'autre part l'influence de l'objet à extraire sur ce même environnement avec, par exemple, l'apparition d'ombres. Il est possible d'avoir un aperçu des différentes techniques existantes dans [Pic04]. Parmi celles-ci, on notera le modèle de la bibliothèque *OpenCV* [Int01]. Il prend en compte les différences de luminosité sur une période de temps donnée. A cette fin, N images de l'arrière plan sont prises sur un intervalle de temps fixé, puis pour chaque pixel on va calculer la somme $S(x,y)$ des valeurs ainsi que leur carré $Sq(x,y)$. La déviation standard de chaque pixel est alors obtenue en posant $\sigma(x,y) = \sqrt{\frac{Sq(x,y)}{N} - \frac{S(x,y)}{N}^2}$. Un pixel $p(x,y)$ est alors considéré comme n'appartenant pas à l'arrière plan si $|m(x,y) - p(x,y)| > C\sigma(x,y)$ avec $m(x,y) = \frac{S(x,y)}{N}$ et C une constante donnée (pouvant valoir 3 pour correspondre à la règle empirique des *trois sigmas*).

Une approche proposée par Horpasert *et al.* [HHD99] permet, en plus de détecter les éléments n'appartenant pas à l'arrière plan, de définir les zones où se produisent localement des changements d'illumination causés par les ombres, les hautes lumières ou des variations de luminosité extérieures. Le modèle se base sur les différences de chrominance CD et de luminosité α par rapport à la couleur que devrait avoir un pixel donné. Une image de référence est créée en se basant sur les valeurs a_i et b_i associées à chaque pixel i. a_i est la variation de α et b_i la variation de CD sur N images et sont obtenus en utilisant la moyenne des couleurs de chaque pixel sur N images ainsi que sa déviation standard. Pour chaque nouveau pixel, les valeurs $\widehat{\alpha_i}$ et $\widehat{CD_i}$ sont calculées et représentent respectivement la distorsion sur la luminosité et sur la chrominance. On peut alors en déduire à quelle catégorie un pixel appartient (voir un exemple de résultat sur la figure 84) :

$$\begin{cases} \text{objet si} & \widehat{CD_i} > \tau_{CD} \\ \text{arriere plan si} & \widehat{\alpha_i} < \tau_{\alpha 1} et \widehat{\alpha_i} > \tau_{\alpha 2} \\ \text{ombre si} & \widehat{\alpha_i} < 0 \\ \text{haute lumiere} & sinon \end{cases}$$

Dans notre cas, nous sommes dans les conditions particulières d'une utilisation en salle de réalité virtuelle, ce qui signifie par exemple que l'éclairage est contrôlé. Les variations de luminosité sont effectivement négligeables et l'ombrage est relativement faible. Nous avons donc cherché à effectuer une extraction de l'arrière plan pouvant être suffisamment rapide, mais ne conservant que les zones de l'image utiles à la construction de notre *visual hull*. Lors de la première étape, un nombre donné d'images de l'arrière plan sont enregistrées et une moyenne est calculée pour chaque pixel. La raison de ces multiples captures est de pallier la faiblesse des capteurs CCD présents sur des caméras de type *webcam* qui ont tendance à être très sensibles (granularité).

La seconde étape va consister à déterminer si un pixel appartient à l'arrière plan ou

FIG. 84: Application de la méthode d'extraction d'arrière plan avec le sujet recherché en bleu, les ombres en rouge et les hautes lumières en vert. *Copyright : Horpasert et al.*

au sujet dont on cherche à obtenir la silhouette. Pour évaluer si un pixel donné appartient à la silhouette, la distance Euclidienne entre sa couleur et sa référence est calculée pour déterminer leur différence de chrominance dans l'espace RVB. Si cette différence est supérieure à une constante τ donnée, alors le pixel est considéré comme appartenant à la silhouette. Dans le cas où cette différence resterait encore faible (de l'ordre de deux fois la constante τ) nous ajoutons un test sur la différence de teinte à l'aide d'un produit scalaire. Si on obtient une valeur proche de 1 alors les deux couleurs seront considérées comme ayant la même teinte et appartiennent donc à l'arrière plan. Comme cette partie est un traitement local, il est alors tout à fait possible d'exploiter l'accélération graphique via l'utilisation des *shaders* comme présenter dans le listing 3.1.

Listing 3.1: Code *GLSL* d'extraction de la silhouette

```
1  uniform sampler2D texture;
2  uniform sampler2D meantexture;
3  uniform float normlim;
4  uniform float difflim;
5
6  void main(void)
7  {
8
9    vec3 color = texture2D(texture,gl_TexCoord[0].xy).rgb;
10   vec3 mean  = texture2D(meantexture,gl_TexCoord[0].xy).rgb;
11
12   float l = length(mean-color);
13   if(l < difflim )
14     discard;
```

```
15      else{
16        if(l < difflim*2.0){
17          float a = (dot(color,mean)/(length(mean)*length(color)));
18           if( a > normlim )
19                  discard;
20         }
21       }
22      gl_FragColor = vec4(color,1.0);
23  }
```

Afin d'affiner la détection de la silhouette et retirer les pixels parasites, nous allons appliquer, sur le résultat et à l'aide du *GPU*, les opérateurs de morphologie d'ouverture (composition d'une érosion[22] et d'une dilatation[23]) et de fermeture (composition d'une dilatation et d'une érosion). L'ouverture va permettre d'éliminer les amas de pixels isolés tandis que la fermeture va contribuer à remplir des zones appartenant à la silhouette alors définies comme vides. À l'instar de la méthode d'extraction du paragraphe précèdent, il est possible d'appliquer ces opérations en utilisant un programme sur les fragments, ce qui va augmenter la rapidité des calculs.

La dernière étape de l'extraction de la silhouette consiste à créer la polyligne délimitant son contour. Pour cela, nous utilisons la bibliothèque *OpenCV*, dont les calculs se basent sur la méthode de Teh et Chin [TC89] qui permet de trouver les points dominants sur une courbe. Un algorithme de simplification du contour, issu des travaux de Douglas et Pleucker [DP73], est ensuite appliqué au résultat.

3.1.2 Reconstruction du modèle virtuel

La construction du modèle virtuel, via la technique des *visual hulls*, nécessite la création des volumes composés par les polylignes des silhouettes et ayant pour centre la position de la caméra dans le référentiel choisi. Cette dernière est déterminée en utilisant la relation de l'équation 2.6 qui nous permet de poser que $\mathbf{t} = -\mathtt{R}\mathbf{C}$ et donc, on peut en déduire la position de la caméra est $\mathbf{C} = -\mathtt{R}^{-1}\mathbf{t}$.

Pour trouver les volumes, l'étape suivante va consister à «déprojeter» chacun des points 2D $\mathbf{p}$ d'une polyligne afin d'obtenir un de ses équivalents $\mathbf{p}'$ dans l'espace 3D. En effet la solution de l'application de la projection inverse $\mathtt{P}^{-1}$ n'est pas unique et se présente sous la forme d'une ligne. Un point p' peut alors être calculé avec $\mathbf{p}' = \mathtt{P}^{-1}\mu\mathbf{p} + \mathbf{C}$. μ est une constante qui va alors définir la distance du point $\mathbf{p}'$ par rapport au centre de projection. Plus cette distance sera grande, plus la taille du volume le sera également (cas typique d'une projection à l'infini). Un maillage constitué de triangles est finalement créé

[22] Pour un élément structurant M et une image I données, l'érosion se définit comme $ERO_M(I) = \{(x,y) \| M_{(x,y)} \subset I\}$

[23] Pour un élément structurant M et une image I données, la dilatation se définit comme $DIL_M(I) = \{(x,y) | M_{(x,y)} \cap I = \emptyset\}$

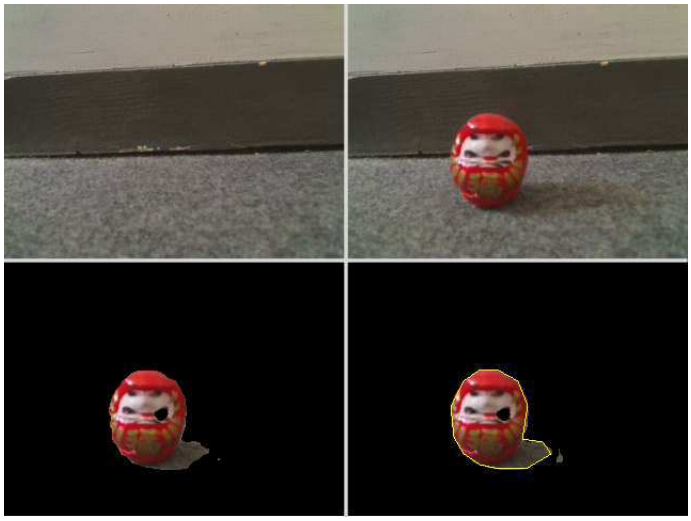


FIG. 85: Aperçu des différentes étapes du procédé d'extraction de la silhouette. Une petite partie des ombres est toutefois inclus dans le résultat.

en utilisant l'ensemble des points $\mathbf{p}'$ et le point $\mathbf{C}$ comme base. Un exemple de volumes obtenus par cette méthode est présenté dans la figure 86.

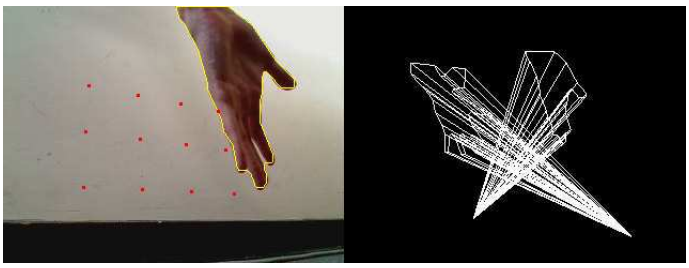


FIG. 86: Présentation de la création de deux volumes à l'aide des silhouettes calculées depuis deux caméras

Le *visual hull* est obtenu en appliquant la technique de *CSG* (*Constructive Solid Geometry*) et des opérations binaires peuvent ainsi être utilisées entre les différents volumes. Il existe deux types de mise en application de cet algorithme : une va calculer un nouveau maillage après application de l'opérateur et une autre se repose uniquement sur un résultat visuel. Cette dernière catégorie fait pleinement usage des possibilités de la carte graphique comme le présentent McReynolds et Blythe dans [MB97]. Ils proposent d'appliquer les opérations binaires en utilisant le *stencil buffer*[24] qui permet de ne conserver que certaines parties de la zone d'affichage. Par exemple, un *ET* binaire peut être obtenu de la manière suivante :

[24]Espace mémoire de la carte graphique dans lequel il est possible d'associer une valeur à chaque pixel et d'appliquer des tests sur ces derniers en fonction d'une valeur de référence.

Algorithm 3.1.1: STENCIL CSG()

Soit a et b deux objets
Initialiser le *depth buffer* avec a
Désactiver l'écriture dans le *depth buffer*
Incrémenter le *stencil buffer* partout où les faces avant de b peuvent être dessinées
Décrémenter le *stencil buffer* partout où les faces arrière de b peuvent être dessinées
Désactiver le test de profondeur
Afficher a où le stencil buffer est différent de 0
Faire la même chose en inversant a et b

Cette méthode fonctionne parfaitement lorsqu'il s'agit de simplement afficher le résultat d'un *CSG*, mais ne permet cependant pas d'interaction avec les autres éléments virtuels (physique, ombres,...). Pour ces raisons nous préférons l'utilisation d'une méthode *CSG* permettant la création d'un nouveau maillage. Notre choix s'est porté sur l'utilisation de la bibliothèque *GTS* (*GNU Triangulated Surface*) qui offre la possibilité d'appliquer un certain nombre d'algorithme sur des surfaces et plus particulièrement des opérations binaires.

3.1.3 Le rendu du *visual hull*

La phase de rendu du *visual hull* consiste à définir les textures qui vont s'appliquer sur le maillage de l'avatar, en considérant les informations récupérées depuis les différents points de vue. D'une manière générale le but est de projeter l'ensemble des textures sur le modèle en se fondant sur les matrices de projection calculées pour chacune des caméras comme expliqué dans [Eve01].

La technique la plus «naturelle» est de calculer l'angle qu'il y a entre la normale d'un sommet et le rayon issu de ce même sommet et dirigé vers une des caméras. Le point de vue retenu pour définir la couleur à projeter est alors celui qui forme l'angle le plus petit. Cependant cette démarche ne tient pas compte des occlusions ; cela signifie qu'un sommet non visible depuis une caméra se verra quand même attribuer la texture produite depuis cette dernière en raison de l'angle de faible valeur.

Une solution permettant de résoudre ce problème d'occlusion, et que nous avons retenu pour faire notre rendu du *visual hull*, a été proposé par Li *et al.* [LMS03] et Sawhney [SAK+02]. Elle repose sur le même principe que celui de création d'ombres par *shadow mapping* décrite dans la section 3.2.2 de ce chapitre. Brièvement décrite, une carte de profondeur du modèle géométrique est récupérée depuis chacune des caméras. En utilisant un programme sur les *shaders*, il est ensuite possible de transformer chaque sommet pour récupérer le résultat dans le repère du point de vue désiré et dans celui des multiples caméras. En utilisant ces dernières informations, une comparaison est effectuée entre la profondeur du fragment p_f et la valeur associée dans la carte de profondeur p_t récupérée depuis une caméra précise. Si la différence $p_f - p_t$ est négative, alors il faut considérer

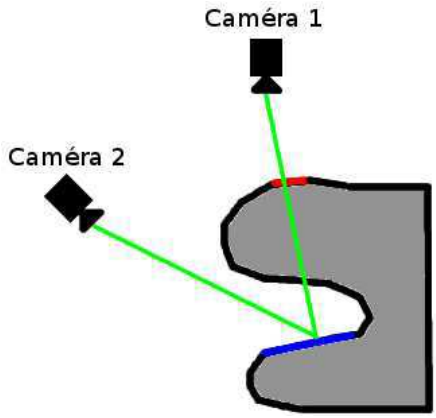


FIG. 87: Il est possible d'utiliser les informations de la caméra 1 pour définir la texture de la facette rouge. Même si la normale de la facette bleue semble définir la caméra 1 comme source d'information, il faut utiliser une autre caméra en raison du problème de visibilité.

que le sommet n'est pas visible depuis cette caméra. Dans le cas contraire, la texture produite depuis cette caméra peut être employée pour colorier le fragment. Étant donné que la solution n'est pas unique nous terminons la sélection de la texture à appliquer en reconsidérant la caméra comme celle la mieux orientée parmi celles retenues.

Outre la principale limite causée par la résolution des textures qui a tendance à réduire la qualité du résultat, il faut préciser que le nombre de passes est équivalent au nombre de caméras pour créer les cartes de profondeur (produites via un *FBO*) plus une pour le rendu. Pour le moment le nombre maximal de textures qu'il est possible de combiner pour faire le rendu est limité à huit dans la version actuelle d'*OpenGL*, ce qui signifie que le nombre maximal de caméras est tout autant limité. Le code du *shader* associé à cette étape de rendu est décrit dans le listing 3.2.

Listing 3.2: Code adapté du *shadow mapping* pour résoudre le problème des occlusions

```
1  // Vertex shader
2  uniform mat4 cameramatrix[8];
3  uniform vec3 camerapos[8];
4  varying vec4 pvertex[8];
5  varying float angles[8];
6  varying vec4 vertexpos;
7
8  void main(void)
9  {
10  vertexpos = vec4(gl_Vertex);
11
12  vec3 normal   = vec3(gl_NormalMatrix * gl_Normal);
13  vec3 vertexpos = (gl_ModelViewMatrix * gl_Vertex).xyz;
14
15  for(int i = 0; i < 8; ++i){
```

```
16      pvertex[i] = cameramatrix*gl_Vertex;
17      vec3 vectmp = normalize((gl_ModelViewMatrix*
18                          vec4(camerapos[i],1.0)).xyz - vertexpos);
19      angles[i] = acos(dot(vectmp,normal));
20    }
21
22     gl_Position = ftransform();
23  }
24
25  /*****************************************************/
26
27  // Fragment shader
28  uniform sampler2D shadowmap[8];
29  uniform sampler2D textures[8];
30  uniform mat4 texturesmat[8];
31
32  varying vec4 pvertex[8];
33  varying float angles[8];
34  varying vec4 vertexpos;
35
36  void main(void)
37  {
38   int cpt = 0;
39   float shadow;
40   vec4 color[8];
41   int cameraid[8];
42   for(int i = 0; i < 8; ++i){
43     float shadow = texture2D(shadowmap[i],pvertex[i].xy).r;
44     if(shadow <= pvertex[i].z - 0.001 ){
45       color[cpt] = texture2DProj(textures[i],
46                                  texturesmat[i]*vertexpos);
47       cameraid[cpt] = i;
48       ++cpt;
49     }
50   if(cpt > 0){
51     vec4 finalcolor = color[0];
52     float angle = angles[cameraid[0]];
53     for(int i = 1; i < cpt; ++i){
54       if(angles[cameraid[i]] < angle){
55         finalcolor = color[i];
56         angle = angles[cameraid[i]];
57     }
58     gl_FragColor = vec4(finalcolor,1.0);
59   }
60  }
```

La phase de rendu est assez coûteuse du fait des nombreuses passes nécessaire à l'ob-

tention du résultat surtout lorsqu'elle est additionnée au temps nécessaire pour le calcul du *visual hull*. Pour cette raison nous avons décidé de dissocier ces deux étapes en déportant les calculs sur deux machines. Le résultat de la reconstruction est alors transmis du premier ordinateur vers le second en utilisant une connexion réseau et un échange de données assuré par la bibliothèque VRPN [RMTHS+01].

Nous profitons de cette première machine pour enregistrer le modèle géométrique de l'avatar dans un *Vertex Buffer Object* ce qui permet en plus, de définir un format de stockage pour la transmission des données vers la seconde machine, d'offrir une structure de données efficace et intéressante pour effectuer rapidement les multiples rendus de l'avatar. Nous profitons également de cette machine pour calculer les différentes cartes de profondeur. Pour éviter d'avoir trop de textures à transférer nous avons choisi d'utiliser le canal alpha des images extraites des caméras pour stocker la valeur de profondeur. De cette manière le nombre de textures est divisé par deux.

3.2 Exploitation

La fonction de l'avatar créé par le biais de la méthode de reconstruction des *visual hulls* est d'accroître le sentiment de présence d'un utilisateur. Nous cherchons à refléter l'existence de ce dernier dans l'environnement virtuel de manière naturelle. Dans la vie courante plusieurs indices visuels nous permettent d'être conscient de notre présence. On trouve, tout d'abord, la possibilité d'observer directement les différentes parties de notre corps comme nos pieds ou nos mains. Nous pouvons inclure l'interaction qui peut se produire avec d'autres individus, car ils peuvent interagir avec nous via un dialogue ou une action. Ensuite il est possible de se percevoir en observant notre reflet sur différents types de surfaces que peuvent être un miroir, du métal ou simplement une surface translucide ou liquide. Enfin, un autre signe sont les ombres que nous projetons involontairement sur les objets et qui vont s'animer en accord avec les mouvements que nous effectuons.

Dans cette partie, nous allons nous intéresser aux deux derniers cas de notre autoperception. Le rejet du premier cas est motivé par le système de réalité virtuelle employé qui est organisé autour d'un écran de grande taille. Utiliser un avatar pour représenter virtuellement les parties de notre corps que nous pouvons observer serait alors contradictoire car nous observerions en même temps, par exemple, notre bras réel et notre bras virtuel. Le rejet du second correspond à une volonté de rechercher à améliorer l'immersion par l'utilisation d'indices visuels passifs plutôt que la gestion d'acteurs virtuels. Nous allons donc présenter, dans un premier temps, une méthode pour que l'utilisateur puisse se voir dans le monde virtuel par l'utilisation de surfaces réfléchissantes. Puis nous présenterons la manière avec laquelle il est possible de générer une ombre pour marquer différemment la présence de l'utilisateur dans l'environnement. Enfin nous étudierons d'autres pistes d'exploitation de l'avatar.

3.2.1 Perception via les reflets

création d'une surface réflective

Une surface réfléchissante, comme le miroir, est une surface qui a la particularité de renvoyer l'ensemble ou une partie des rayons lumineux, dits incidents, qui l'atteignent. Les rayons réémis, dits réfléchis, vont former avec la normale à la surface, un angle équivalent à celui formé entre la normale et le rayon incident. De cette manière, et comme l'illustre la figure 88, un objet invisible depuis une position du point de vue donnée peut le devenir par l'intermédiaire du miroir. Ce processus pourrait être décrit d'une manière différente. En effet, ce que nous voyons au travers du miroir peut être assimilé à une modification de la position du point de vue qui consisterait à l'application d'une symétrie ayant pour axe la surface du miroir. De la même façon nous pouvons considérer que le contenu du reflet d'un miroir peut être obtenu en appliquant la même symétrie sur l'ensemble des objets composants l'environnement. Pour résumer, les objets observés sur une surface réfléchissante sont dépendants d'un axe de symétrie et de la position du point de vue. Diverses méthodes existent en synthèse d'images pour générer des surfaces de ce type. Étant destiné à une application de réalité virtuelle, nous nous bornerons aux algorithmes qui produisent un résultat en temps réel (le lancer de rayons est par exemple exclu).

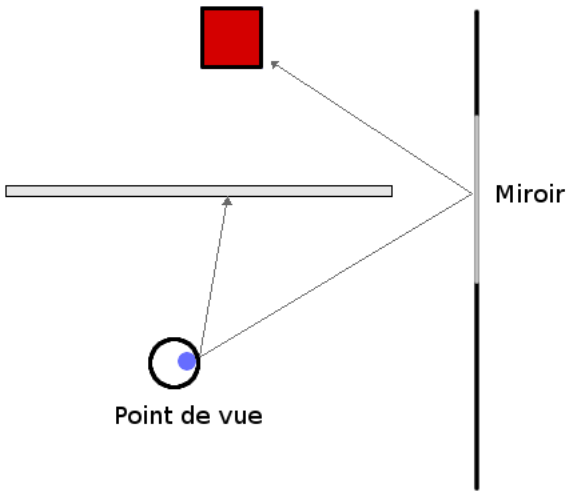


FIG. 88: Un objet occulté par un autre peut devenir visible au travers de l'utilisation d'un miroir

La technique la plus courante de création du reflet, issu d'une surface plane, consiste à utiliser le *stencil buffer*[25] [McR00]. L'objectif est de dupliquer la scène via une symétrie axée sur cette surface mais dont la visibilité est réduite à la zone du miroir. Le miroir peut alors être décrit comme un espace vide à travers lequel il est possible d'observer l'inverse

[25]Espace mémoire de la carte graphique permettant d'effectuer des tests complémentaires sur les fragments.

de la scène.

La création du contenu du miroir va se faire en deux passes. La première va consister à uniquement afficher le miroir de manière à initialiser le *stencil buffer* avec une valeur particulière. Dans la seconde, la scène va subir une symétrie qui se résume à l'application d'une matrice de transformation R [Gol90] définie par :

$$\mathtt{R} = \begin{bmatrix} 1-2V_x^2 & -2V_xV_y & -2V_xV_z & 2(P\cdot V)V_x \\ -2V_yV_x & 1-2V_y^2 & -2V_yV_z & 2(P\cdot V)V_y \\ -2V_zV_x & -2V_zV_y & 1-2V_z^2 & 2(P\cdot V)V_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

avec P un point du miroir et V sa normale. Notons que si la symétrie est appliquée simplement de cette manière alors les objets qui sont derrière le miroir vont se retrouver devant et être ajoutés au résultat (figure 89). Cette limite est corrigée en définissant un plan dit de *clipping*[26]. Il devient alors possible de restreindre l'affichage aux seuls objets, qui après l'application de la symétrie, se trouveront derrière le plan du miroir.

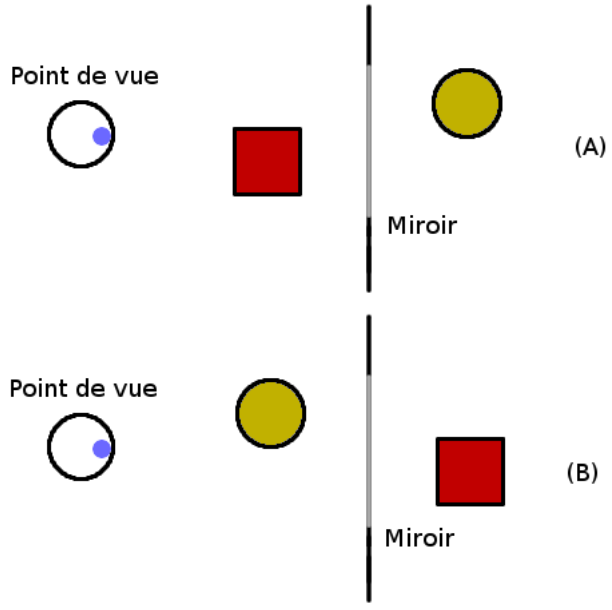


FIG. 89: illustration du problème rencontré lors de l'application de la symétrie, axée sur le miroir, sur l'ensemble de la scène *(A)*. Sans prendre en compte l'opération de *Clipping*, les objets précédemment situés derrière le miroir deviennent visibles *(B)*

Finalement le rendu n'est effectué que dans les zones du miroir en se servant d'un test sur les valeurs contenues dans le *stencil buffer* : tous les fragments en dehors de l'aire

[26]Bornage du résultat de l'affichage à l'espace situé à l'avant ou à l'arrière d'un plan.

définie par le miroir sont éliminés. L'algorithme de rendu peut alors se résumer à :

Algorithm 3.2.1: REFLECTION PAR STENCIL BUFFER()

Désactiver l'écriture dans le buffer des couleurs
Définir la fonction du test du *stencil* de manière à l'initialiser à 1 lorsqu'un fragment est rendu
Dessiner uniquement le miroir
Réactiver l'écriture dans le buffer des couleurs
Dessiner la scène
Appliquer la matrice de transformation R
Définir le plan de *clipping*
Activer le test du *stencil*
Dessiner la scène
Désactiver le test du *stencil*
Désactiver le *clipping*

Un miroir créé de cette manière va paraître trop parfait, car il va refléter l'image exacte de la scène. Pour ajouter un peu plus d'imperfections, une surface exactement identique est créée et placée au même niveau que le miroir. Une texture avec un canal de transparence est plaquée dessus et va permettre de donner un peu plus de «réalisme». Le fait de placer deux surfaces, exactement au même endroit, entraîne des conflits au moment du test de profondeur qui se manifeste par un ensemble d'artefacts sur celles-ci. En utilisant la technique du *polygon offset*[27], le résultat sera affiché correctement.

Le principal défaut qui nous contraint à utiliser une autre méthode, provient du fait que cette dernière monopolise le *stencil buffer* qui est un espace mémoire unique sur la carte graphique (il n'est possible de l'utiliser qu'une seule fois pour un seul et même rendu). Or, beaucoup d'autres techniques requièrent l'emploi du *stencil buffer*, ce qui peut poser certains problèmes par exemple lors du rendu de la scène pour définir le contenu du miroir. Il est également important de souligner que l'effet de réflexion sur des surfaces semi-transparentes, comme le verre, est dans ce cas assez complexe et qu'il faut des passes supplémentaires pour obtenir cet effet.

Les faiblesses de cette méthode nous apparaissent importantes de part de notre volonté d'afficher des ombres ou d'utiliser des vitres comme support potentiel de réflexion de l'avatar de l'utilisateur. Nous avons donc opté pour une méthode de création de miroir qui fait appel aux textures [MB05]. L'algorithme se déroule cette fois-ci en deux passes : lors de la première, on applique sur la scène la matrice de symétrie en conservant l'utilisation du plan de *clipping*. Le résultat du rendu est stocké dans une texture par le biais des *FBO*. La seconde passe va consister à afficher la scène normalement en plaquant correcte-

[27] Technique qui permet de modifier la valeur de profondeur des fragments au moment du test de profondeur.

ment la texture de la passe précédente sur la surface définissant le miroir. Cette action est rendue possible en utilisant la génération automatique des coordonnées de texture d'*OpenGL* et les transformations calculées à partir des matrices de projection et *modelview*. Cela garantit que la bonne zone de la texture s'affichera sur le miroir. L'utilisation d'une texture plutôt qu'un simple rendu symétrique permet l'ajout d'un canal alpha sur la surface du miroir. Cela offre la possibilité de pouvoir moduler la transparence de la surface réfléchissante à notre guise pour, par exemple, créer un reflet sur une vitre de fenêtre. L'algorithme servant à générer une surface peut être détaillé de la manière suivante :

Algorithm 3.2.2: REFLECTION PAR TEXTURE()

Activer le *FBO*
Définir le plan de *clipping*
Appliquer la matrice de transformation R
Dessiner la scène
Désactiver le *clipping*
Désactiver le *FBO*
Dessiner la scène
Calculer la matrice de transformation de la texture
Dessiner le miroir en plaquant dessus le résultat du *FBO*

Notons que pour gérer les réflexions multiples, dû par exemple à la présence de plusieurs miroirs, il est nécessaire d'effectuer des passes supplémentaires. Il faut dans un premier temps, attribuer une texture à chacun des miroirs présents dans la scène, puis dans une seconde partie, réitérer l'algorithme jusqu'à obtenir le niveau de réflexion souhaité. La figure 92 illustre les résultat obtenus avec cette méthode. On peut constater que l'utilisateur a la possibilité de se voir dans la scène au travers des surfaces réfléchissantes qui présentent un moyen naturel d'intégrer l'avatar dans le monde virtuel.

Position du point de vue

Comme cela avait déjà été mentionné au début de la précédente section, la position de l'observateur est importante dans un système composé de surfaces réfléchissantes. Il est donc nécessaire de connaître à chaque instant où se trouve la tête de l'utilisateur. A priori la connaissance de la direction dans laquelle l'utilisateur regarde n'a pas d'importance car la scène affichée est considérée comme statique. En effet, dans notre cas, nous considérons le support de projection comme une «fenêtre» vers le monde virtuel, ce qui implique qu'il existe une cohérence entre l'espace dans lequel évolue l'utilisateur et celui qu'on tente de lui faire accepter.

L'autre intérêt de connaître la position du point de vue de l'utilisateur réside dans la correction des mouvements pseudoscopiques que l'on rencontrent avec les images stéréoscopiques [NB03]. Ces mouvements se caractérisent par un léger décalage de l'image qui

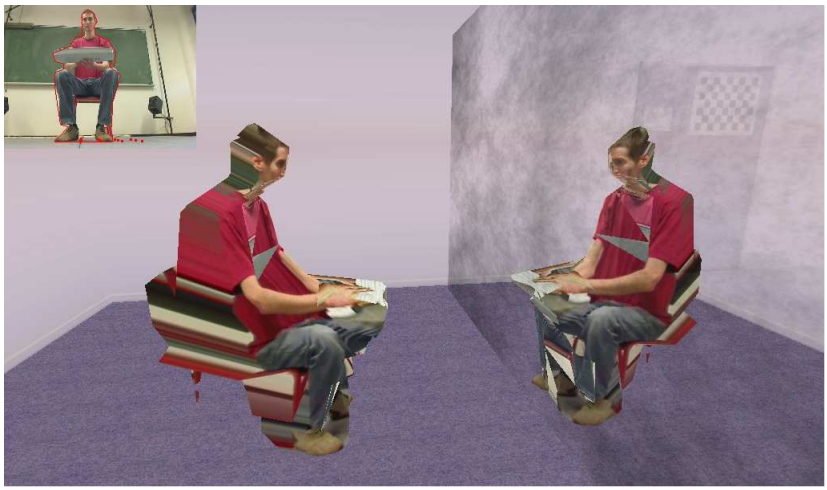


FIG. 90: La surface semi réfléchissante permet à l'utilisateur de s'observer dans l'environnement virtuel. La flaiblesse du rendu sur les flanc est causée par une position frontale des différentes caméras.

accompagne les déplacements du point de vue de l'observateur. Cet effet provient du fait que la fusion des images produit du relief ou de la profondeur en des points de l'espace dépendant du point de vue, comme illustré sur la figure 91. Une solution consiste à adapter la perspective de l'image affichée, en considérant la position de la tête de l'observateur. Contrairement à la fois précédente, les calculs de ces modifications tiennent compte de la direction du regard car ils sont appliqués à la manière des calculs d'une image stéréoscopique et qui requièrent de connaître la position des deux yeux de l'observateur.

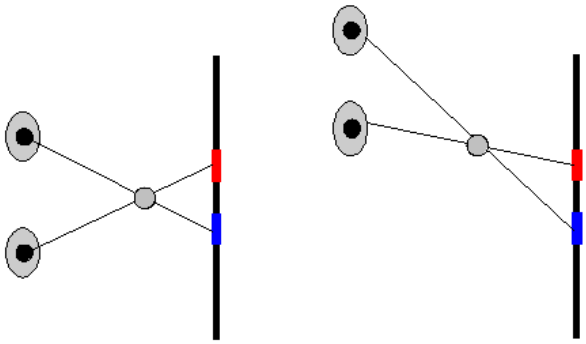


FIG. 91: Lorsque l'utilisateur bouge sa tête en observant une image stéréoscopique fixe, il va avoir une impression de décalage dont l'origine est liée aux mouvements pseudoscopiques

Chez l'être humain il est possible d'établir les proportions entre la taille d'un individu et la hauteur à laquelle se situent ses yeux. A cette fin, nous nous basons sur le modèle d'esthétisme d'un homme ordinaire, couramment utilisé en dessin ou sculpture et qui décrit comme un canon de sept fois et demi la tête. Concrètement, la taille d'un homme

est considérée comme étant proportionnelle à sept fois et demi la taille de sa tête. De la même manière les yeux sont considérés comme étant au milieu de la tête. Finalement il est possible d'en déduire que proportionnellement à la taille, les yeux sont situés à environ 93,5% de la taille totale. De la même manière, on peut en déduire que les yeux sont situés à 6,5% de la taille totale d'un individu depuis le sommet de sa tête.

Concrètement, il devient possible en tenant compte de ces considérations, de définir la position du point de vue de l'utilisateur à partir des informations obtenues depuis le *visual hull*. Il est relativement simple de récupérer le point le plus élevé de l'avatar pour en déduire la position des yeux. Il est donc nécessaire dans un premier temps de calculer la taille totale de l'utilisateur. Pour obtenir ce résultat, il est demandé à ce dernier de se tenir debout, de manière normale, pendant un court, instant avec les bras le long du corps de façon à construire un avatar à partir duquel il est possible de récupérer le point le plus élevé et le point le plus bas. La taille est alors obtenue en effectuant une différence entre ces deux valeurs. Finalement, on en déduit la position des yeux e comme étant :

$$e = \mathrm{p}_{\mathrm{max}} - (size * 0.65) \tag{3.2}$$

avec $size$ la taille de l'utilisateur et p_max le point de l'espace le plus élevé sur l'avatar considéré comme étant le sommet de la tête (ne prenant par conséquent pas la levée des bras en compte). La distance entre le sommet de la tête et le niveau des yeux est toujours fixe. De ce fait, même si l'utilisateur est accroupi ou entrain d'effectuer un saut, la position de ses yeux pourra être considérée comme correcte.

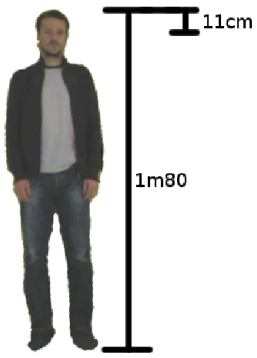


FIG. 92: La position du point peut être éstimée comme étant, par rapport au sommet de la tête, à environ 6,5% de la taille de l'utilisateur.

Pour obtenir plus de précision sur la détection de la position des yeux, il est nécessaire d'approfondir la reconnaissance de la posture de l'avatar en lui associant, par exemple, un squelette. De cette manière, nous pouvons obtenir plus efficacement une évaluation de l'orientation de la tête et ainsi la direction approximative du regard. Un ensemble de méthodes permettant d'obtenir un squelette a été présenté dans la section 2.4 de ce

chapitre.

Enfin, il est important de préciser que dans ce cas nous obtenons un *tracking* de l'observateur sans avoir à utiliser des marqueurs. Le principal avantage étant que l'utilisateur n'a plus besoin de porter une quelconque interface, comme des capteurs optiques ou magnétiques, ou alors de se préparer en revêtant une combinaison particulière. Le fait de se passer de l'ensemble des éléments perturbateurs va pleinement aider l'utilisateur à créer un sentiment de présence.

3.2.2 Exploitation de l'éclairage virtuel

Outre la possibilité de se servir de l'avatar pour générer des reflets de l'utilisateur via des miroirs virtuels afin d'augmenter le sentiment de présence chez l'utilisateur, il existe un autre moyen de souligner une appartenance de celui-ci à l'environnement virtuel. Le principe consiste à faire interagir l'avatar avec l'ambiance lumineuse de la scène virtuelle [SUC95]. A l'instar d'une surface réfléchissante qui va retransmettre les différents mouvements de l'utilisateur, il est possible d'obtenir le même résultat en s'appuyant sur la génération d'ombres qui vont permettre de naturellement retranscrire les mouvements de l'utilisateur dans l'espace virtuel. Pour cela nous allons tenir compte de la position des différentes sources de lumière de la scène pour produire les ombres provoquées par la présence de l'avatar.Deux techniques sont principalement utilisées en synthèse d'images pour créer des ombres.

La première, dont le nom est *shadow mapping* [Wil78], est couramment utilisée dans des applications graphiques à cause de la capacité de la méthode à produire rapidement un résultat. Le principe est globalement de comparer les profondeurs des fragments entre le point de vue de la caméra et la même scène, vue depuis la source de lumière. La marche à suivre consiste à récupérer dans une première passe la carte de profondeur de la scène rendue depuis la source de lumière dans une texture en utilisant l'extension *Frame Buffer Objets*. Durant cette même phase, les valeurs des matrices *modelview* et de projection sont récupérées et multipliées pour obtenir une matrice M . Dans la seconde passe, la scène est rendue normalement par l'intermédiaire d'un programme sur les *shaders*. Ce dernier va nous permettre de décider si un fragment doit être considéré comme étant dans l'ombre ou pas. Pour cela, chaque sommet est transformé via la matrice M pour obtenir, une fois homogénéisé, sa profondeur depuis la source de lumière. Enfin, il suffit de comparer cette valeur p à celle stockée dans la texture t calculée lors de la première passe, pour en déduire si le résultat doit être «ombré» : si $p > t$ alors le fragment doit être dans l'ombre, dans le cas contraire le fragment est laissé en l'état. Le principal désavantage de cette technique est le stockage du résultat de la première passe dans une texture ce qui va avoir pour conséquence de produire un résultat généralement de basse résolution qui se caractérise par un crénelage des contours des ombres mais certaines adaptations de l'algorithme permettent toutefois d'en limiter l'effet, comme dans [SD02].

La seconde technique se nomme *shadow volume* [Cro77, Car00, BS03] et se fonde sur

la génération d'un volume avec une origine se situant au niveau de la source de lumière et dont chaque rayon passe par les points composant le contour (silhouette d'un objet vu de puis la source de lumière) d'un objet dont on cherche à projeter l'ombre. La suite consiste à employer le *stencil buffer* (en désactivant l'écriture dans le *buffer* de couleur et de profondeur) que l'on va incrémenter lorsqu'une face avant est affichée et décrémenter lorsqu'il s'agit d'un face arrière. Si au final la valeur stockée dans le stencil buffer est différente de 0 alors cela signifie que le fragment est situé dans l'ombre. La plus grande contrainte de la méthode réside dans la génération des volumes «projetant» l'ombre dans la scène et dont la discrétisation des objets va influencer la résolution des ombres. Au contraire, le principal avantage est que le calcul ne s'applique que sur des objets bien précis ce qui permet d'accélérer les calculs de rendu. L'algorithme peut être résumé de la manière suivante :

Algorithm 3.2.3: SHADOW VOLUME()

Désactiver l'écriture dans les *buffers* de couleur et de profondeur
Activer l'affichage des faces avant
Définir l'opération du stencil avec une incrémentation lors de l'échec du test de profondeur
Afficher le volume
Activer l'affichage des faces arrières
Définir l'opération du *stencil* avec une décrémenation lors de l'échec du test de profondeur
Afficher le volume

L'évaluation de la silhouette de l'objet occultant, vu depuis la source de lumière, est une étape consommatrice en temps de calcul. Elle nécessite, en effet, de parcourir l'ensemble des triangles visibles depuis la source de lumière. On en déduit la silhouette, en extrayant les arêtes de ces triangles qui n'apparaissent qu'une seule fois. Une nouvelle fois, cette étape peut être accélérée en faisant appel aux *shaders*. Il est possible dans les *geometry shaders* de récupérer les triangles adjacents à celui qui est en cours de traitement. Ainsi, on peut en déduire qu'une arête appartient à la silhouette si, d'une part la facette est visible depuis la source de lumière et d'autre part si son voisin, avec lequel elle partage cette arête, n'est par contre pas visible. Nous pouvons également profiter de cette étape pour générer l'extrusion à l'infini des arêtes de la silhouette. Enfin, il est possible sur les dernières cartes graphiques, d'utiliser l'extension OpenGL *EXT_stencil_two_side*. Elle permet, en une seule passe, de traiter les faces avants et arrières avec des fonctions de *stencil* différentes. La combinaison de ces deux optimisations permet donc de passer de trois passes des calculs à une seule.

Nous avons opté pour l'utilisation de cette dernière méthode, car elle reflète notre volonté d'obtenir un résultat suffisamment crédible pour l'utilisateur. Le *shadow volume* permet ainsi de projeter, de manière simple et efficace, l'ombre de l'avatar sur les différents objets de la scène avec un résultat visuel satisfaisant. Au contraire, l'emploi du *shadow mapping* va plutôt s'utiliser sur les objets secondaires composant la scène. Un résultat de

l'application de cette méthode est présenté dans la figure 93.

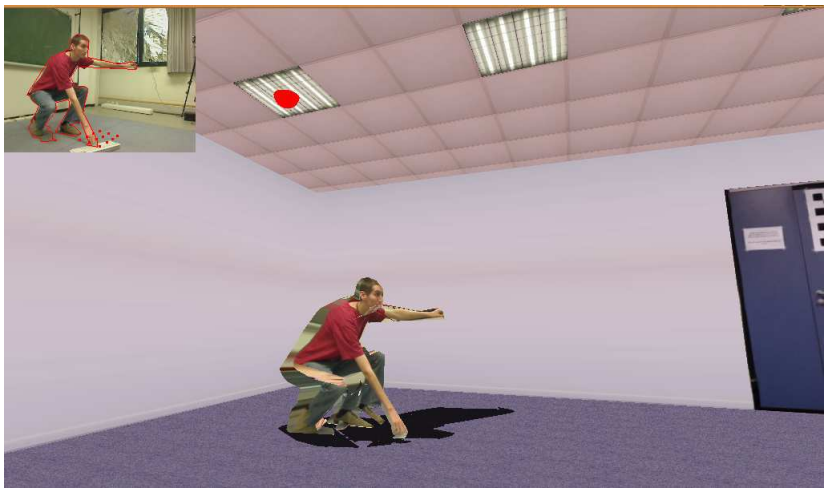


FIG. 93: Exemple de l'influence d'une lumière virtuelle sur l'avatar. La position de la source de lumière est représentée par une sphère rouge.

Un autre effet de l'éclairage virtuel est son influence sur l'avatar. Lorsque l'avatar est construit par le biais de la méthode des *visual hulls*, on va calculer le rendu à lui appliquer en fonction des images reçues depuis les différentes caméras. Cela implique que le résultat va tenir compte des multiples influences lumineuses présentes tout autour de l'utilisateur et qui ne correspondent pas forcément à celles présentes dans la partie virtuelle. Concrètement l'avatar va être par exemple affiché avec une texture comportant des zones ombrées alors qu'elles ne le devraient pas. A l'heure actuelle, il existe très peu d'algorithme permettant la soustraction de ces différents effets de lumière sur une image (ombre, lumière spéculaire,..). On pourra quand même citer les travaux de Fredembach et Finlayson [FF06] et de Xu *et al.* [XQJ06] qui proposent tous les deux de bons résultats mais dont les performances ne sont pas adaptées au temps réel. Nous envisageons ainsi d'obtenir des résultats similaires en adaptant ces algorithmes sur la carte graphique afin d'accroître ses performances.

3.2.3 Autres possibilités

En plus de refléter la présence de l'utilisateur au moyen de miroirs virtuels ou de la projection d'ombres, l'avatar peut être utilisé pour interagir avec l'environnement virtuel. Même si le modèle géométrique de l'avatar n'est pas directement visible par l'utilisateur, il n'en représente pas moins une intégration de ce dernier dans la scène virtuelle. Nous pouvons donc faire en sorte d'utiliser cette enveloppe pour en déduire de possibles interactions.

Une première approche consiste à employer un moteur physique tels que *ODE*, *Newton* ou *Havok* pour créer des interactions réalistes en détectant les collisions avec les objets

composants la scène virtuelle. Des actions de ce genre ne permettent toutefois pas d'en déduire la volonté propre de l'utilisateur d'effectuer telle ou telle action.

Une autre possibilité d'exploitation de l'avatar est la traduction de la volonté de l'utilisateur à se déplacer dans l'environnement virtuel. Cela nécessite de détecter les mouvements des jambes de l'utilisateur et de reconnaître la description d'une marche. D'une manière plus simple, il est aussi possible de détecter les points de contact entre les pieds de l'avatar et le sol. Si il y a deux points de contact, alors il n'y a pas de mouvement, par contre si un enchaînement de points de contact uniques se produit on peut alors en déduire qu'il y a un mouvement. Comme évoqué précédemment, ce genre de solution n'intègre toutefois pas la notion de volonté et dans ce cas il est difficile de pouvoir distinguer une envie de se déplacer de celle de rester sur place.

La détection des pas peut enfin être utilisée pour accroître l'immersion par stimulation du sens auditif. Dans cette situation, il est possible d'associer un son à chaque fois que le modèle 3D de l'utilisateur entre en contact avec le sol du monde virtuel. Ainsi, cette utilisation participe à aider l'utilisateur à mieux percevoir l'expérience à laquelle il participe.

3.3 Résultats

L'objectif de cette section est de décrire notre démarche pour créer une méthodologie visant à vérifier si nos hypothèses concernant l'utilisation d'un avatar sont correctes. A l'heure actuelle, la création de cette étude est en cours afin de pouvoir proposer un protocole d'évaluation du sentiment de présence pertinent.

3.3.1 Evaluation du sentiment de présence

Pour évaluer le sentiment de présence au cours d'une expérience de réalité virtuelle, il existe plusieurs méthodes dont les plus courantes se fondent sur l'utilisation d'un questionnaire. La mesure de la présence peut se faire sur un certain nombre de données, comme l'explique Insko dans [TR+03]. Il évoque en premier lieu une mesure subjective de la présence en justifiant que ce sentiment est propre à chaque personne et que l'une des manière les plus efficace de l'évaluer est de poser une série de questions à l'utilisateur.

Parmi les questionnaires qui existent, le plus utilisé est celui de Winter et Singer [WS98] qui vise à être le plus généraliste possible pour pouvoir s'adapter aux différentes situations dans lesquelles peut se dérouler une expérience de réalité virtuelle. Plus précisément, ils ont ajoutés un coefficient à chaque question des facteurs qui sont associés à des catégories comme le «contrôle», les «sens», la «distraction» (au sens d'éléments extérieurs perturbateurs) et le «réalisme». Slater n'étant pas en accord avec cette mesure de la présence [Sla99], il a par la suite proposé, en compagnie de Usoh *et al.* sa propre vision

de ce que devait être un questionnaire d'évaluation du sentiment de présence [UCAS00]. Les questions cette fois sont définies selon trois catégories qui sont le «*being there*» (ou «être présent»), la proportion selon laquelle l'expérience prend le pas sur le réel et enfin la proportion selon laquelle l'environnement paraît exister. L'utilisateur soumis à six questions doit alors y répondre en utilisant une échelle de valeurs reflétant ses impressions. Un dernier type de questionnaire remarquable est celui proposé par Lessiter *et al.* [LFKD00]. Celui-ci a la particularité d'être adapté à n'importe quelle situation d'expérimentation. Ainsi, soixante trois questions ont été déduites en association avec quinze catégories comprenant l'implication, la crédibilité ou les effets négatifs sur un grand échantillon de personnes.

Concernant les autres mesures de la présence, il y a celles relatives au comportement de l'utilisateur c'est-à-dire qu'une phase d'observation de ce dernier est organisée en le plaçant dans des conditions expérimentales particulières. Il est également possible de mesurer les paramètres physiologiques de l'utilisateur pour détecter, par exemple, des variations dans les battements de son cœur qui pourraient être corrélés au déroulement de l'expérience.

A côté de cela, Slater a proposé une toute autre vision de l'évaluation du sentiment de présence qu'il décrit de la manière suivante :

«*[...] Est-il possible de faire en sorte que les gens fuient un feu virtuel ? Si ce n'est pas le cas, quel que soit le score obtenu lors d'un questionnaire sur la présence, ils n'ont pas montré de présence. S'ils avaient le sentiment d'appartenir à l'environnement virtuel, avec le feu, ne voudraient-ils pas fuir ? [...]*» [SFFD+07]

Autrement dit, pour Slater le sentiment de présence n'a lieu que si nos réactions sont les mêmes que dans la vie réelle. D'une certaine manière, il décrit ce que serait un système de réalité virtuelle parfait qui pour le moment n'est pas disponible.

3.3.2 Proposition et Discussion

Pour évaluer si notre hypothèse d'améliorer le sentiment de présence par l'intégration de façon naturelle de l'utilisateur dans l'environnement virtuel, nous devons établir un protocole pertinent permettant de nous éclairer sur son état vis-à-vis de l'expérience. Cette phase, encore en étude, va principalement reposer sur notre version de la définition de la présence que nous avons donné dans le premier chapitre. Nous évoquions alors que la présence nécessite de faire appel aux quatre piliers que sont l'immersion, l'interaction, le maintien de la boucle action-perception et la création d'émotions, mais pas forcément avec les mêmes proportions.

De notre point de vue, notre proposition respecte une partie de l'immersion de l'utilisateur dans l'environnement virtuel. L'immersion ne peut être pas considérée comme complète car elle ne fait que stimuler la vue de l'utilisateur et dans une moindre mesure

l'ouïe si nous considérons la création d'un son pour marquer son déplacement. Toutefois, comme cela a été montré dans le chapitre 1, la vue est notre principal outil de perception ce qui fait que sa stimulation en réalité virtuelle est très importante. Ainsi, la possibilité offerte à l'utilisateur de pouvoir se voir ou se percevoir dans l'univers virtuel va alors pleinement participer à l'immerger dans l'expérience.

L'utilisation de la méthode des *visual hulls* pour calculer une copie de l'utilisateur a l'avantage de pouvoir obtenir un bon résultat visuel en temps réel. Cette dernière particularité est obligatoire pour maintenir la boucle action-perception qui peut être décrite comme la nécessité de faire en sorte que lorsque l'utilisateur effectue une action la réaction lui apparaisse de manière naturelle. Dans notre cas un affichage en temps réel de l'avatar de l'utilisateur assure ce maintien. En effet, si l'utilisateur décide de bouger une partie de son corps, il faut alors que le résultat soit immédiatement répercuté dans l'environnement virtuel.

En ce qui concerne les émotions, il faut considérer qu'il est assez exceptionnel pour un observateur d'un univers 3D de pouvoir se voir à l'intérieur. Pour exemple, les jeux vidéo n'offrent que très ponctuellement cette possibilité comme sur la *Playstation 2* avec l'*Eyetoy*. Faire en sorte que l'utilisateur puisse se voir va donc très probablement l'inciter à se sentir plus impliqué voire développer chez lui un intérêt qui va l'aider à plus facilement à accepter les imperfections qui subsistent.

Le cas de l'interaction est un peu plus délicat. Pour agir sur l'environnement virtuel, il est toujours possible de se servir de l'avatar pour détecter des collisions et déclencher des événements physiques. Mais cela reste assez limité dans le sens où cela ne refléte pas forcément ses réelles intentions. De plus, le retour tactile du monde virtuel sur l'utilisateur n'est réalisable qu'au travers d'interfaces et donc impossible en l'état. Une solution serait alors d'étudier d'éventuelles pistes dirigées vers le pseudo-haptique.

Concrètement l'évaluation du sentiment de présence devra être effectuée en tenant compte de l'ensemble de ces facteurs au travers d'une série de questions posées à l'utilisateur. Chaque pilier sera donc abordé tout en proposant des variations au niveau de l'affichage pour déterminer si nos choix ont bien été intégrés de manière naturelle et mettre en évidence des bénéfices sur le sentiment de présence.

Conclusion

La réalité virtuelle est un domaine encore en pleine expansion notamment parce que la plupart des disciplines dont elle fait usage ne cessent de s'améliorer. Une finalité, qui consiste à faire en sorte que l'utilisateur se sente présent dans l'environnement virtuel, requiert de contribuer de manière satisfaisante à quatre piliers. Nous les avons définis comme étant l'immersion, l'interaction, la conservation de la boucle action-perception et les émotions. Notre objectif était donc de proposer différentes approches qui pourraient permettre d'améliorer le sentiment de présence lors d'une expérience de réalité virtuelle. Étant donné la diversité des possibilités offertes par les interfaces de réalité virtuelle, nous avons choisi de nous concentrer sur la restitution visuelle. Plus particulièrement, nous avons placé l'ensemble de nos travaux dans le cadre d'une utilisation sur écran immersif (grande taille) ou autostéréoscopique.

Nous avons commencé par proposer deux outils destinés à mieux aider l'utilisateur à percevoir les environnements virtuels. Notre première approche concerne la transcription sur la carte graphique de l'algorithme permettant de générer des images stéréoscopiques. Traditionnellement le relief est obtenu en produisant deux rendus d'une même scène mais depuis deux points de vue légèrement décalés. Ce procédé ne tient donc pas compte des nombreux points communs qui existent entre les sommets d'un point de vue à l'autre, ce qui provoque des calculs redondants. Depuis l'apparition des *geometry shaders*, il devient possible de manipuler les primitives géométriques après l'étape du calcul sur les sommets (illumination, coordonnées, ...). Malgré quelques limitations qui seront corrigées grâce aux prochaines versions des cartes graphiques, notre méthode permet en une seule passe de générer une image stéréoscopique, voire plusieurs images pour une utilisation destinée, par exemple, aux écrans auto-stéréoscopiques. De plus, notre méthode permet d'accélérer les temps de rendu en évitant de doubler certaines étapes des calculs et en employant pleinement les avantages offerts par les cartes graphiques.

Sur le sujet de la perception, nous nous sommes également intéressés à trouver un moyen de limiter les effets causés par l'observation prolongé d'images stéréoscopiques. Nombreux sont les utilisateurs de systèmes de réalité virtuelle qui subissent des fatigues ou même des maux de tête après avoir passé une période plus ou moins longue à regarder de telles images. Ces problèmes sont généralement liés à la difficulté à fusionner les deux images à cause de leur écart important. Notre solution est de détourner le regard de l'utilisateur de ces zones considérées comme problématiques en appliquant dessus un flou de profondeur de champ calculé en temps réel par le biais des *shaders*. Ce flou, produit par

la présence d'une lentille dans un système optique, permet de rendre certains éléments moins attirants pour l'œil tout en leur conservant un aspect naturel en raison de notre habitude à le percevoir dans les films ou sur des photographies.

L'autre remarque sur les environnements virtuels concerne la tendance que possède la plupart des utilisateurs à laisser leur regard vagabonder dans l'image à la recherche d'un point d'intérêt. Nous avons donc proposé d'aider l'utilisateur à focaliser son attention en réutilisant le flou de profondeur de champ : le regard de l'utilisateur a alors plus facilement tendance à se concentrer vers les zones de l'image que nous définissons comme importantes. Pour cela, nous avons décrit un certain nombre de processus qui peuvent décrire ces zones. Nous trouvons ainsi les méthodes automatiques qui se fondent sur les caractéristiques des objets affichés comme leur taille, leur mouvement, leur couleur... Nous avons également présenté une méthode utilisant des points de contrôle qui vont aider à déterminer où la zone de netteté doit être positionnée en fonction de la situation de l'utilisateur dans l'environnement. Un cas d'utilisation est par exemple celui de la visite d'un musée virtuel. Enfin, nous avons proposé un système de scripts qui permettent de contrôler le flou tout le long d'un scénario.

Dans la dernière partie de ce mémoire nous avons présenté une méthode destinée à améliorer l'immersion de l'utilisateur en lui faisant croire qu'il est bien présent dans l'environnement virtuel. Pour cela nous avons conservé notre choix axé sur une focalisation sur l'aspect visuel et par conséquent nous avons opté pour la «virtualisation» de l'utilisateur sous la forme d'un avatar. Parmi toutes les méthodes existantes qui permettent d'obtenir, à l'aide de plusieurs caméras, un rendu de l'utilisateur depuis n'importe quel point de vue, nous avons sélectionné la technique des *visual hulls*. Cette dernière offre de nombreux avantages dont celui de pouvoir générer un modèle en 3D en temps réel dont la qualité est suffisante pour l'exploitation que nous désirons en faire.

Divers cas d'utilisation d'un avatar en réalité virtuelle ont déjà été présentés mais sont soit uniquement adapté pour être employés avec un *HMD*, soit affichés sous la forme d'un clone de l'utilisateur sans que ce dernier n'arrive réellement à s'identifier à lui comme étant son propre prolongement. Notre apport est d'effectuer l'intégration de l'avatar d'une manière plus naturelle en nous servant de supports virtuels qui permettent de refléter la propre image de l'utilisateur. Nous utilisons ainsi des réflexions sur des miroirs ou des vitres pour que l'utilisateur puisse s'apercevoir dans l'environnement virtuel. Nous utilisons aussi l'influence de l'éclairage de la scène virtuelle sur l'avatar pour générer des ombres que l'utilisateur peut observer effectuant les mêmes gestes que lui. Enfin nous proposons de profiter de l'avatar pour effectuer le *tracking* de l'utilisateur afin, par exemple, de corriger des problèmes de perception liés aux mouvements pseudoscopiques induits par les images stéréoscopiques.

L'évaluation du sentiment de présence appliqué à nos diverses solutions, nécessite de mettre en œuvre un protocole permettant de tirer le maximum d'informations de l'expérience subie par un utilisateur. Cette partie requiert un nombre de connaissances importantes en ce qui concerne les sciences cognitives afin d'obtenir un processus d'évaluation

pertinent. Cette étape nécessaire est notre principal axe pour la continuité de notre travail parce que dans un premier temps nos hypothèses doivent être correctement validées et ensuite nous devons avoir à notre disposition un tel outil pour nos futurs travaux sur l'amélioration du sentiment de présence.

Dans la continuité de ces travaux, nous envisageons d'améliorer le rendu des ombres de l'avatar en essayant d'utiliser les informations issues des différentes caméras. Nous pourrons alors employer un *shadow mapping* basé sur l'image plutôt que sur un modèle géométrique. De plus, nous proposerons de diversifier les surfaces réfléchissantes en les étendant à des surfaces non planaires ou nécessitant une fonction de réflexion plus complexe (comme fondée sur la $BRDF$). Ensuite, pour améliorer la création du sentiment de présence, nous voulons développer nos travaux sur l'utilisation des écrans autostéréoscopiques en réalité virtuelle. Enfin, en parallèle, nous continuerons nos recherches sur l'amélioration de l'immersion en proposant l'utilisation d'images panoramiques en relief issues d'images de synthèse ou de caméras omnidirectionnelles.

Bibliographie

[3-D] ColorCode 3-D : Colorcode 3-d. Site internet. http ://www.colorcode3d.com/.

[AAK71] YI Abdel-Aziz et HM Karara : Direct linear transformation into object space coordinates in close-range photogrammetry. *Proc. Symp. Close-Range Photogrammetry*, 18, 1971.

[AB92] Steve Aukstakalnis et David Blatner : *Silicon Mirage ; The Art and Science of Virtual Reality*. Peachpit Press, Berkeley, CA, USA, 1992.

[ABF+04] J. Allard, E. Boyer, J.S. Franco, C. Ménier et B. Raffin : Marker-less Real Time 3D Modeling for Virtual Reality. *Immersive Projection Technology*, 2004.

[ABM+97] Maneesh Agrawala, Andrew C. Beers, Ian McDowall, Bernd Fröhlich, Mark Bolas et Pat Hanrahan : The two-user responsive workbench : support for collaboration through individual views of a shared space. *In SIGGRAPH '97 : Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 327–332, 1997.

[AFT03] Bruno Arnaldi, Philippe Fuchs et Jacques Tisseau : *Le traité de la réalité virtuelle*, chapitre 1, page 9. Les Presses de l'Ecole des Mines de Paris, 2003.

[AH93] Stephen J. Adelson et Larry F. Hodges : Stereoscopic ray-tracing. *The Visual Computer*, 10(3):127–144, 1993.

[Bar99] Barco : Consul compact, dual screen, l-shaped projection table. Brochure, 1999. http ://www.barco.com/projection_systems/downloads/consul.pdf.

[Bau75] B.G. Baumgart : A polyhedron representation for computer vision. *AFIPS National Computer Conference*, 44:589–596, 1975.

[BB97] Edmond Boyer et M.-O. Berger : 3d surface reconstruction using occluding contours. *International Journal of Computer Vision*, 22(3):219–233, 1997.

[BBA+04] R.M. Banos, C. Botella, M. Alcaniz, V. Liano, B. Guerrero et B. Rey : Immersion and emotion : Their impact on the sense of presence. *CyberPsychology & Behavior*, 7(6):734–741, 2004.

[BBP+98] C. Botella, RM Baños, C. Perpiñá, H. Villa, M. Alcañiz et A. Rey : Virtual reality treatment of claustrophobia : a case report. *Behaviour Research and Therapy*, 36(2):239–246, 1998.

[BBX95] Chandrajit L. Bajaj, Fausto Bernardini et Guoliang Xu : Automatic reconstruction of surfaces and scalar fields from 3d scans. *In SIGGRAPH '95 : Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 109–118, New York, NY, USA, 1995. ACM.

[BC08] Luca Ballan et Guido Maria Cortelazzo : Marker-less motion capture of skinned models in a four camera set-up using motion flow and silhouettes. *In 4th International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT'08)*, juin 2008. electronic version (8 pp.).

[BD00] E. Borovikov et L. Davis : A distributed system for real-time volume reconstruction. *In CAMP '00 : Proceedings of the Fifth IEEE International Workshop on Computer Architectures for Machine Perception (CAMP'00)*, page 183, Washington, DC, USA, 2000. IEEE Computer Society.

[BDD+00] B. Buxton, L. Dekker, I. Douros, T. Vassilev et G. Street : Reconstruction and Interpretation of 3D Whole Body Surface Images. *Scanning 2000 Proceedings*, 2000.

[BF03] Edmond Boyer et Franco Franco : A hybrid approach for computing visual hulls of complex objects. *cvpr*, 01:695, 2003.

[BFB+98] N.A. Borghese, G. Ferrigno, G. Baroni, A. Pedotti, S. Ferrari et R. Savare : Autoscan : a flexible and portable 3d scanner. *Computer Graphics and Applications, IEEE*, 18(3):38–41, May/Jun 1998.

[BLK+07] A. Buttari, P. Luszczek, J. Kurzak, J. Dongarra et G. Bosilca : Cop3 : a rough guide to scientific computing on the playstation 3. Rapport technique UT-CS-07-595, Innovative Computing Laboratory, University of Tennessee Knoxville, 2007.

[BM02] W. Boehler et A. Marbs : 3D Scanning Instruments. *Proceedings of the CIPA WG*, 6, 2002.

[Bou] Alexis Bousiges : Arcade-history. Site internet. http ://www.arcade-history.com.

[BS03] S. Brabec et H.P. Seidel : Shadow Volumes on Programmable Graphics Hardware. *Computer Graphics Forum*, 22(3):433–440, 2003.

[BvdHHV00] John Bastian, Anton van den Hengel, Ken Hawick et Francis Vaughan : Modelling the Perceptual Effects of lens Distortion via an Immersive Virtual Reality System. Rapport technique DHPC-086, 2000.

[Car00] J. Carmack : John carmack on shadow volumes. *Available from developer. nvidia. com*, 2, 2000.

[CCAD01] Nathan Chia, Richard Cant et David Al-Dabass : New anti-aliasing and depth of field techniques for games graphics. *In GAME-ON*, pages 115–120, 2001.

[CGH05] Fabrice Caillette, Aphrodite Galata et Toby Howard : Real-time 3-d human body tracking using variable length markov models. *British Machine Vision Conference*, 2005.

[Cla93] Constance CLASSEN : *Worlds of Sence : Exploring the sens in history and across cultures*. Routledge, 1993.

[CNSD+92] Carolina CRUZ-NEIRA, Daniel J. SANDIN, Thomas A. DEFANTI, Robert V. KENYON et John C. HART : The cave : audio visual experience automatic virtual environment. *Commun. ACM*, 35(6):64–72, 1992.

[Col96] Robert T. COLLINS : A space-sweep approach to true multi-image matching. *cvpr*, 00:358, 1996.

[con] Web 3D CONSORTIUM : X3d and related specifications. http ://www.web3d.org/x3d/specifications/.

[CPC84] Robert L. COOK, Thomas PORTER et Loren CARPENTER : Distributed ray tracing. *SIGGRAPH Comput. Graph.*, 18(3):137–145, 1984.

[Cro77] F.C. CROW : Shadow algorithms for computer graphics. *ACM SIGGRAPH Computer Graphics*, 11(2):242–248, 1977.

[Cur98] Brian Lee CURLESS : *New methods for surface reconstruction from range images*. Thèse de doctorat, Stanford, CA, USA, 1998.

[Cut97] J.E. CUTTING : How the Eye Measures Reality and Virtual Reality. *Behavior research methods instruments and computers*, 29:27–36, 1997.

[Dag05] Mehmet DAGDELEN : *Restitution des stimuli inertiels en simulation de conduite*. Thèse de doctorat, École des Mines de Paris, 2005.

[Dem04] J. DEMERS : *Depth of Field : A Survey of Techniques*, chapitre 23, pages 375–390. Addison-Wesley Professional, 2004.

[DH02] J. DIFEDE et H.G. HOFFMAN : Virtual Reality Exposure Therapy for World Trade Center Post-traumatic Stress Disorder : A Case Report. *CyberPsychology & Behavior*, 5(6):529–535, 2002.

[Dod05] N. A. DODGSON : Autostereoscopic 3d displays. *Computer*, 38(8):31–36, 2005.

[DP73] D.H. DOUGLAS et T.K. PEUCKER : Algorithms for the reduction of the number of points required to represent a digitizer line or its caricature. *Cartographica : The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[DSG94] T.M.H. DIJKSTRA, G. SCHONER et C.C.A.M. GIELEN : Temporal stability of the action-perception cycle for posturalcontrol in a moving visual environment. *In Experimental Brain Research*, pages 477–486, 1994.

[dSNB08a] François de SORBIER, Vincent NOZICK et Venceslas BIRI : Accelerated stereoscopic rendering using gpu. *In 16th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2008 (WSCG'08)*, ISBN 978-80-86943-16-9, février 2008.

[dSNB08b] François de SORBIER, Vincent NOZICK et Venceslas BIRI : Gpu rendering for autostereoscopic displays. *In 4th International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT'08)*, juin 2008. electronic version (7 pp.).

[DYB98] P.E. DEBEVEC, Y. YU et G.D. BORSHUKOV : Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping. *Eurographics Rendering Workshop*, 98:105–116, 1998.

[Dye01] C.R. DYER : Volumetric Scene Reconstruction from Multiple Views. *Foundations of Image Understanding*, pages 469–489, 2001.

[Eis06] Peter EISERT : *Reconstruction of Volumetric 3D Models*, chapitre 8, pages 133–150. 2006.

[Ele06] Philips ELECTRONICS : Wowvx for amazing viewing experiences. *Philips 3D solutions*, 2006.

[Eng04] W.F. ENGEL : *ShaderX2 : Shader Programming Tips & Tricks with DirectX 9*. Wordware Publishing, 2004.

[Eve01] Cass EVERITT : Projective texture mapping. white paper, 2001.

[FB03] Jean-Sébastien FRANCO et Edmond BOYER : Exact polyhedral visual hulls. *In Proceedings of the Fourteenth British Machine Vision Conference*, pages 329–338, September 2003. Norwich, UK.

[Feh03] C. FEHN : A 3d-tv approach using depth-image-based rendering (dibr). *In Visualization, Imaging, and Image Processing*, 2003.

[FF06] C. FREDEMBACH et G. FINLAYSON : Simple shadow remova. *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, 1:832–835, 0-0 2006.

[Fis99] Scott S. FISHER : *Virtual Environments, Personal Simulation, & Telepresence*, pages 101–111. MIT Press, 1999.

[FK03] R. FERNANDO et M.J. KILGARD : *The Cg Tutorial : The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2003.

[FLB06] Jean-Sébastien FRANCO, Marc LAPIERRE et Edmond BOYER : Visual shapes of silhouette sets. *In Proceedings of the 3rd International Symposium on 3D Data Processing, Visualization and Transmission, Chapel Hill (USA)*, 2006.

[FMHR87] Scott S. FISHER, M. MCGREEVY, J. HUMPHRIES et W. ROBINETT : Virtual environment display system. *In SI3D'86 : Proceedings of the 1986 workshop on Interactive 3D graphics*, pages 77–87, New York, NY, USA, 1987. ACM.

[FPR+01] Maurizio FORTE, Eva PIETRONI, Claudio RUFA, Angela BIZZARRO, Alessandro TILIA et Stefano TILIA : Dvr-pompei : a 3d information system for the house of the vettii in opengl environment. *In VAST '01 : Proceedings of the 2001 conference on Virtual reality, archeology, and cultural heritage*, pages 307–314, New York, NY, USA, 2001. ACM.

[Fus04] Joaquim M. FUSTER : Upper processing stages of the perception-action cycle. *Cognitive Sciences*, 8(4):143–145, 2004.

[GA93] Meister Eduard GRÖLLER et Pietro ACQUISTO : *A distortion camera for ray tracing*, page 14. Elsevier Science Publishers, avril 1993.

[GHF86] J. GOLDFEATHER, J.P.M. HULTQUIST et H. FUCHS : Fast constructive-solid geometry display in the pixel-powers graphics system. *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 107–116, 1986.

[GKG04] Indra GEYS, Thomas P. KONINCKX et Luc Van GOOL : Fast interpolated cameras by combining a gpu based plane sweep with a max-flow regularisation algorithm. *In 3DPVT '04 : Proceedings of the 3D Data Processing, Visualization, and Transmission, 2nd International Symposium*, pages 534–541, Washington, DC, USA, 2004. IEEE Computer Society.

[GM03] B. GOLDLUCKE et M. MAGNOR : Real-time microfacet billboarding for free-viewpoint video rendering. *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, 3:III–713–16 vol.2, Sept. 2003.

[Gol90] Ronald GOLDMAN : Matrices and transformations. pages 472–475, 1990.

[GVG04] I. GEYS et L.J. VAN GOOL : Extended view interpolation by parallel use of the GPU and the CPU. *Optical Materials*, 5665:96–107, 2004.

[GWN+03] M. GROSS, S. WURMLIN, M. NAEF, E. LAMBORAY, C. SPAGNO, A. KUNZ, E. KOLLER-MEIER, T. SVOBODA, L. VAN GOOL, S. LANG *et al.* : blue-c : A Spatially Immersive Display and 3D Video Portal for Telepresence. *ACM International Conference Proceeding Series*, 39:9–10, 2003.

[HA90] Paul HAEBERLI et Kurt AKELEY : The accumulation buffer : hardware support for high-quality rendering. *In SIGGRAPH '90 : Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 309–318, New York, NY, USA, 1990. ACM.

[HA99] Milton P. HUANG et Norman E. ALESSI : *Presence as an emotional experience*, pages 148–153. IOS Press, 1999.

[HDD+92] Hugues HOPPE, Tony DEROSE, Tom DUCHAMP, John MCDONALD et Werner STUETZLE : Surface reconstruction from unorganized points. *Computer Graphics*, 26(2):71–78, 1992.

[Hei62] Morton L. HEILIG : United states patent 3050870 : Sensorama simulator, 1962.

[Hei88] Adam HEILBRUN : Jaron lanier : A vintage virtual reality interview. Interview en ligne, 1988. http ://www.jaronlanier.com/vrint.html.

[HHD99] Thanarat HORPRASERT, David HARWOOD et Larry S. DAVIS : A statistical approach for real-time robust background subtraction and shadow detection. *In ICCV Frame-Rate WS*, 1999.

[HLCC07] Sébastien HILLAIRE, Anatole LÉCUYER, Rémi COZOT et Géry CASIEZ : Effets de flou visuel pour la navigation en environnements virtuels en vue à

la première personne. *In Actes de l'Association Française d'Informatique Graphique (AFIG)*, pages 83–90, 2007.

[HLGB03] J.M. Hasenfratz, M. Lapierre, J.D. Gascuel et E. Boyer : Real-time capture, reconstruction and insertion into virtual world of human actors. *Vision, Video and Graphics Conference*, 2003.

[HSS97] Wolfgang Heidrich, Philipp Slusallek et Hans-Peter Seidel : An image-based model for realistic lens systems in interactive computer graphics. *In Proceedings of the conference on Graphics interface '97*, pages 68–75, Toronto, Ont., Canada, Canada, 1997. Canadian Information Processing Society.

[Hug95] Howard Hugues : Seeing, hearing and smelling the world. rapport sur la science biomédicale, 1995.

[HZ04a] R. I. Hartley et A. Zisserman : *Multiple View Geometry in Computer Vision*, chapitre 6, pages 239–261. Cambridge University Press, ISBN : 0521540518, second édition, 2004.

[HZ04b] R. I. Hartley et A. Zisserman : *Multiple View Geometry in Computer Vision*, chapitre 7, pages 178–194. Cambridge University Press, ISBN : 0521540518, second édition, 2004.

[HZ04c] R. I. Hartley et A. Zisserman : *Multiple View Geometry in Computer Vision*, chapitre 9, pages 239–261. Cambridge University Press, ISBN : 0521540518, second édition, 2004.

[HZP06] Thomas Hübner, Yanci Zhang et Renato Pajarola : Multi-view point splatting. *In GRAPHITE '06 : Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, pages 285–294, 2006.

[HZP07] Thomas Hübner, Yanci Zhang et Renato Pajarola : Single-pass multi-view rendering. *IADIS International Journal on Computer Science and Information Systems 2007*, 2(2):122–140, 2007.

[ICP04] Theodore Mayer III, Todd A. Chaney et Lawrence S. Paul : United states patent 6690337 : Multi-panel video display, 2004.

[Int01] Intel : Open source computer vision library : Reference manual, 2001.

[iO] iO : Sensistive wall. Site internet. http ://www.sensitivewall.com/.

[Ive03] Fredrick E. Ives : United states patent 725567 : Parallax stereogram and process for making same, 1903.

[Ive36] Herbert E. Ives : United states patent 2039648 : Camera for making parallax panoramagrams, 1936.

[IYFN05] H. Iwata, H. Yano, H. Fukushima et H. Noma : Circulafloor [locomotion interface]. *Computer Graphics and Applications, IEEE*, 25(1):64–67, 2005.

[IYT06] Hiroo Iwata, Hiroaki Yano et Hiroshi Tomioka : Powered shoes. *In SIGGRAPH '06 : ACM SIGGRAPH 2006 Emerging technologies*, page 28, New York, NY, USA, 2006. ACM.

[JF03] H. Jorke et M. Fritz : INFITEC - a new stereoscopic visualisation tool by wavelength multiplex imaging. *Electronic Displays*, 2003.

[JMY+07] Andrew Jones, Ian McDowall, Hideshi Yamada, Mark Bolas et Paul Debevec : Rendering for an interactive 360° light field display. *ACM Trans. Graph.*, 26(3):40–50, 2007.

[JWS00] X. Ju, N. Werghi et J.P. Siebert : Automatic Segmentation of 3D Human Body Scans. *Proc. Int. Conf. on Computer Graphics and Imaging*, 2000.

[JZ02] Q. Ji et Z. Zhu : Eye and gaze tracking for interactive graphic display, 2002.

[KA06] B. Kaufmann et M. Akil : 3d images compression for multi-view autostereoscopic displays. *Computer Graphics, Imaging and Visualisation, 2006 International Conference on*, pages 128–136, 26-28 July 2006.

[Kap03] Roman Kapusta : Scene reconstruction using structured light. *In 7th Central European Seminar on Computer Graphics*, 2003.

[KGH85] Myron W. Krueger, Thomas Gionfriddo et Katrin Hinrichsen : Videoplace - an artificial reality. *SIGCHI Bull.*, 16(4):35–40, 1985.

[Khr03] Khronos : Opengl extension registry : Gl_arb_occlusion_query. Document en ligne, 2003. http ://oss.sgi.com/projects/ogl-sample/registry/ARB/occlusion_query.txt.

[Khr08a] Khronos : Opengl extension registry : Gl_arb_draw_buffers revision #37. Document en ligne, 2008. http ://www.opengl.org/registry/specs/ARB/draw_buffers.txt.

[Khr08b] Khronos : Opengl extension registry : Gl_ext_framebuffer_object revision #120. Document en ligne, 2008. http ://oss.sgi.com/projects/ogl-sample/registry/EXT/framebuffer_object.txt.

[KLO06] Michael Kass, Aaron Lefohn et John Owens : Interactive depth of field using simulated diffusion on a GPU. Rapport technique #06-01, Pixar Animation Studios, janvier 2006.

[KOF05] Adam G. Kirk, James F. O'Brien et David A. Forsyth : Skeletal parameter estimation from optical motion capture data. *In CVPR 2005*, pages 782–788, juin 2005.

[Kru89] Myron Krueger : *Artificial Reality*. MIT Press, 1989.

[KSV98] Takeo Kanade, Hideo Saito et Sundar Vedula : The 3d room : Digitizing time-varying 3d events by synchronized multiple video streams. Rapport technique CMU-RI-TR-98-34, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 1998.

[Lau91] A. Laurentini : The visual hull : A new tool for contour-based image understanding. *Proc. 7th Scandinavian Conference on Image Analysis*, pages 993–1002, 1991.

[LBE04] Anatole LÉCUYER, Jean-Marie BURKHARDT et Laurent ETIENNE : Feeling bumps and holes without a haptic interface : the perception of pseudo-haptic textures. *In CHI '04 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 239–246, 2004.

[LCK+00] A. LÉCUYER, S. COQUILLART, A. KHEDDAR, P. RICHARD et P. COIFFET : Pseudo-haptic feedback : can isometric input devices simulate force feedback ? *Virtual Reality, 2000. Proceedings. IEEE*, pages 83–90, 2000.

[LD97] Matthew LOMBARD et Theresa DITTON : At the heart of it all : The concept of presence. *Computer Mediated Communication*, 3(2), septembre 1997. http ://www.ascusc.org/jcmc/vol3/issue2/lombard.html.

[Lee07] Johnny Chung LEE : Head tracking for desktop vr displays using the wii remote. Page internet, 2007. http ://www.cs.cmu.edu/ johnny/projects/wii/.

[LFKD00] J. LESSITER, J. FREEMAN, E. KEOGH et J. DAVIDOFF : Development of a new cross-media presence questionnaire : The ITC-Sense of presence. *Presence 2000 Workshop*, pages 27–28, 2000.

[LL06] N. N. LATYPOV et N. N. LATYPOV. : The virtusphere. Site internet, 2006. http ://www.virtusphere.com.

[LLR+06] F. LOTTE, A. LÉCUYER, Y. RENARD, F. LAMARCHE et B. ARNALDI : Classification de données cérébrales par système d'inférence flou pour l'utilisation d'interfaces cerveau-ordinateur en réalité virtuelle. *Actes des premières journées de l'Association Française de Réalité Virtuelle*, pages 767–791, 2006.

[LMS03] M. LI, M. MAGNOR et H.P. SEIDEL : Online accelerated rendering of visual hulls in real scenes. *Journal of WSCG*, 11(2):290–297, 2003.

[LMS04] Ming LI, Marcus MAGNOR et Hans-Peter SEIDEL : Hardware-accelerated rendering of photo hulls. *In* Marie-Paule CANI et Mel SLATER, éditeurs : *The European Association for Computer Graphics 25th Annual Conference EUROGRAPHICS 2004*, volume 23 de *Computer Graphics Forum*, pages 635–642, Grenoble, France, 2004. Blackwell.

[Lok01] B. LOK : Online model reconstruction for interactive virtual environments. *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 69–72, 2001.

[LPG08] Orion Sky LAWLOR, Matthew PAGE et Jon GENETTI : MPIglut : Powerwall programming made easier. *Journal of WSCG*, pages 130–137, February 2008.

[MB97] Tom MCREYNOLDS et David BLYTHE : Programming with opengl : Advanced rendering. SIGGRAPH 97 Course, 1997. http ://www.opengl.org/resources/code/samples/advanced/ advanced97/notes/notes.html.

[MB05] T. MCREYNOLDS et D. BLYTHE : *Advanced Graphics Programming Using OpenGL*. Morgan Kaufmann, 2005.

[MBM01] Wojciech MATUSIK, Chris BUEHLER et Leonard MCMILLAN : Polyhedral visual hulls for real-time rendering. *In Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 115–126, London, UK, 2001. Springer-Verlag.

[MBR+00] Wojciech MATUSIK, Chris BUEHLER, Ramesh RASKAR, Steven J. GORTLER et Leonard MCMILLAN : Image-based visual hulls. *In SIGGRAPH '00 : Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 369–374, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[MBT96] T. MOLET, R. BOULIC et D. THALMANN : A real time anatomical converter for human motion capture. *Eurographics Workshop on Computer Animation and Simulation*, 96:79–94, 1996.

[McR00] Tom MCREYNOLDS : Advanced graphics programming techniques using opengl. Cours, 2000. http ://www.bluevoid.com/opengl/sig00/advanced00/notes/node165.html.

[MFdW07] Yannick MORVAN, Dirk FARIN et Peter H. N. de WITH : Predictive coding of depth images across multiple views. *In Proceedings of SPIE, Stereoscopic Displays and Applications*, 2007.

[MG01] T.B. MOESLUND et E. GRANUM : A survey of computer vision-based human motion capture. *Computer Vision and Image Understanding*, 81(3):231–268, 2001.

[MHK06] T.B. MOESLUND, A. HILTON et V. KRÜGER : A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding*, 104(2-3):90–126, 2006.

[Min95] Marc MINE : Virtual environment interaction techniques. Rapport technique TR95-018, UNC Chapel Hill CS department, 1995.

[MKMA02] N. MALTBY, I. KIRSCH, M. MAYERS et G.J. ALLEN : Virtual Reality Exposure Therapy for the Treatment of Fear of Flying : A Controlled Investigation. *JOURNAL OF CONSULTING AND CLINICAL PSYCHOLOGY*, 70(5):1112–1118, 2002.

[ML00] Jurriaan D. MULDER et Robert Van LIERE : Enhancing fish tank vr. *vr*, 00:91–98, 2000.

[MMM06] Djamel MERAD, Stéphanie METZ et Serge MIGUET : Eye and gaze tracking algorithm for collaborative learning system. *In International Conference on Informatics in Control, Automation and Robotics (IFAC/ICINCO 2006)*, août 2006.

[MvL00] Jurriaan D. MULDER et Robert van LIERE : Fast perception-based depth of field rendering. *In VRST '00 : Proceedings of the ACM symposium on Virtual reality software and technology*, pages 129–133, New York, NY, USA, 2000. ACM.

[NB03] Vincent NOZICK et Venceslas BIRI : Méthode temps réel simple pour la correction des mouvements pseudoscopiques en réalité virtuelle. *In Actes des XVIemes journées de l'AFIG*, pages 231–240, 2003.

[ND93] Jackie NEIDER et Tom DAVIS : *OpenGL Programming Guide : The Official Guide to Learning OpenGL, Release 1.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1993.

[NMA06] V. NOZICK, S. MICHELIN et D. ARQUÈS : Real-time Plane-Sweep with local strategy. *Journal of WSCG*, 14(1-3):121–128, 2006.

[NNC98] MM NORTH, S.M. NORTH et J.R. COBLE : Virtual reality therapy : An effective treatment for phobias. *Virtual environments in clinical psychology and neuroscience : Methods and techniques in advanced patient-therapist interaction*, pages 112–119, 1998.

[NNT07] Christian NITSCHKE, Atsushi NAKAZAWA et Haruo TAKEMURA : Real-time space carving using graphics hardware. *IEICE Transactions*, 90-D(8):1175–1184, 2007.

[NS07] Vincent NOZICK et Hideo SAITO : Multiple view computation for multiple-stereoscopic display. *IEEE Pacific-Rim symposium on image and video technology (PSIVT 2007)*, 4872:399–412, 2007.

[NVI03] NVIDIA : 3d stereo consumer stereoscopic 3d solution. Rapport technique TB-00252-001_v02, NVIDIA, 2003. http ://www.servodata.com.pl/ftp/nvidia/stereo/TB-00271-001_v02-3DStereo.pdf.

[NVI06] NVIDIA : Gpu programming guide version 2.5.0. Document en ligne, 2006. http ://developer.download.nvidia.com/GPU_Programming_Guide/GPU_Programming_Guide.pdf.

[Oko80] T. OKOSHI : Three-dimensional displays. *Proceedings of the IEEE*, 68(5): 548–564, May 1980.

[ope] Open computer vision library. Site internet. http ://sourceforge.net/projects/opencvlibrary/.

[PC82] Michael POTMESIL et Indranil CHAKRAVARTY : Synthetic image generation with a lens and aperture camera model. *ACM Trans. Graph.*, 1(2):85–108, 1982.

[PdS04] Benoit PIRANDA et François de SORBIER : Simulation de flou en synthèse d'images stéréoscopiques pour la réalité virtuelle. *In Poster session of 17eme journées de l'Association Francaise d'Informatique Graphique (AFIG'04)*, pages 43–51, novembre 2004.

[PdSA05] Benoit PIRANDA, François de SORBIER et Didier ARQUES : Simulation of blur in stereoscopic image synthesis for virtual reality. *In Poster session of 13th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2005 (WSCG'05)*, pages 51–52, Plzen, Czech Republic, février 2005.

[PGM03] Christoph PETZ, Bastian GOLDLUECKE et Marcus MAGNOR : Hardware-accelerated autostereogram rendering for interactive 3D visualization. *In* Andrew J. WOODS, John O. MERRITT, Stephen A. BENTON et Mark T.

Bolas, éditeurs : *Stereoscopic Displays and Virtual Reality Systems X*, volume 5006 de *SPIE proceedings*, pages 359–366, Santa Clara, USA, January 2003. The Society for Imaging Science and Technology (IS&T), The International Society for Optical Engineering (SPIE), SPIE.

[Phi06] Philips : Wowvx for amazing viewing experiences. *Philips 3D solutions*, 2006.

[Pic04] M. Piccardi : Background subtraction techniques : a review. *In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, volume 4, 2004.

[PMT05] Cyril Pichard, Sylvain Michelin et Olivier Tubach : Photographic depth of field blur rendering. *In WSCG (Short Papers)*, pages 121–124, 2005.

[Pot87] Michael Potmesil : Generating octree models of 3d objects from their silhouettes in a sequence of images. *Comput. Vision Graph. Image Process.*, 40(1):1–29, 1987.

[Pra06] David Prakel : *composition*. 2006.

[PVGO96a] M. Proesmans, L. Van Gool et A. Oosterlinck : One-shot active 3d shape acquisition. *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*, 3:336–340 vol.3, Aug 1996.

[PvGO96b] M. Proesmans, L.J. van Gool et A.J. Oosterlinck : Active acquisition of 3d shape for moving objects. *Image Processing, 1996. Proceedings., International Conference on*, 3:647–650 vol.3, Sep 1996.

[QRMTHM06] Wen Qi, II Russell M. Taylor, Christopher G. Healey et Jean-Bernard Martens : A comparison of immersive hmd, fish tank vr and fish tank with haptics displays for volume visualization. *In APGV '06 : Proceedings of the 3rd symposium on Applied perception in graphics and visualization*, pages 51–58. ACM, 2006.

[RCM+01] C. Rocchini, P. Cignoni, C. Montani, P. Pingi, R. Scopigno, A. Chalmers et T.M. Rhyne : A low cost 3 Dscanner based on structured light. *EG 2001 Proceedings*, 20:299–308, 2001.

[RDI03] Giuseppe Riva, Fabrizio Davide et Wijnand Ijsselsteijn : *Being There Concepts, effects and measurements of user presence in synthetic environments*. Ios Press, 2003. http ://www.vepsy.com/communication/volume5.html.

[RHHL02] Szymon Rusinkiewicz, Olaf Hall-Holt et Marc Levoy : Real-time 3D model acquisition. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 21(3):438–446, juillet 2002.

[RL01] Szymon Rusinkiewicz et Marc Levoy : Efficient variants of the ICP algorithm. *In Third International Conference on 3D Digital Imaging and Modeling (3DIM)*, juin 2001.

[RMTHS+01] II Russell M. Taylor, Thomas C. Hudson, Adam Seeger, Hans Weber, Jeffrey Juliano et Aron T. Helser : Vrpn : a device-independent,

network-transparent vr peripheral system. *In VRST '01 : Proceedings of the ACM symposium on Virtual reality software and technology*, pages 55–61, New York, NY, USA, 2001. ACM.

[Rob03] D. E. ROBERTS : History of lenticular and related autostereoscopic methods. *Leap Technologies, LLC white paper*, 2003.

[Rok96] P. ROKITA : Generating depth of-field effects in virtual reality applications. *Computer Graphics and Applications, IEEE*, 16(2):18–21, Mar 1996.

[Ros04] R.J. ROST : *OpenGL Shading Language*. Addison-Wesley Professional, 2004.

[RPG+06] A. RIZZO, J. PAIR, K. GRAAP, B. MANSON, P.J. MCNERNEY, B. WIEDERHOLD, M. WIEDERHOLD, J. SPIRA et V.B. INC : A Virtual Reality Exposure Therapy Application for Iraq War Military Personnel with Post Traumatic Stress Disorder : From Training to Toy to Treatment. *NATO SECURITY THROUGH SCIENCE SERIES E HUMAN AND SOCIETAL DYNAMICS*, 6:235, 2006.

[SAK+02] H. S. SAWHNEY, A. ARPA, R. KUMAR, S. SAMARASEKERA, M. AGGARWAL, S. HSU, D. NISTER et K. HANNA : Video flashlights : real time rendering of multiple videos for immersive model visualization. *In EGRW '02 : Proceedings of the 13th Eurographics workshop on Rendering*, pages 157–168, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[SBS02] M. SAINZ, N. BAGHERZADEH et A. SUSIN : Hardware accelerated voxel carving. *1st Ibero-American Symposium in Computer Graphics (SIACG 2002)*, pages 289–297, 2002.

[Sch08] Grant SCHINDLER : Photometric stereo via computer screen lighting for real-time surface reconstruction. *In 3DPVT '08 : Proceedings of the 3D Data Processing, Visualization, and Transmission, 4nd International Symposium*, 2008.

[SCMS01] Gregory G. SLABAUGH, W. Bruce CULBERTSON, Thomas MALZBENDER et Ronald W. SCHAFER : A survey of methods for volumetric scene reconstruction from photographs. *In Volume Graphics*, 2001.

[Sco94] Cary SCOFIELD : $2\frac{1}{2}$*-D Depth of Field Simulation for Computer Animation*, chapitre 1.8, pages 36–38. Academic Press, 1994.

[SD97] Steven M. SEITZ et Charles R. DYER : Photorealistic scene reconstruction by voxel coloring. *In Proc. Computer Vision and Pattern Recognition Conf.*, pages 1067–1073, 1997.

[SD02] M. STAMMINGER et G. DRETTAKIS : Perspective shadow maps. *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 557–562, 2002.

[SFF+00] D.R. SCHIKORE, R.A. FISCHER, R. FRANK, R. GAUNT, J. HOBSON et B. WHITLOCK : High-resolution multiprojector display walls. *Computer Graphics and Applications, IEEE*, 20(4):38–44, Jul/Aug 2000.

[SFFD+07] B. Spanlang, T. Fröhlich, F. F. Descalzo, A. Antley et M. Slater : The making of a presence experiment : Responses to virtual fire. *Presence 2007 -The 10th Annual International Workshop on Presence*, pages 303–307, 2007.

[SHA06] SHARP : 3d lcds. Site internet, 2006. http ://sharp-world.com/products/device/about/technology/lcd-03.html.

[She92] Thomas Sheridan : Musings on telepresence and virtual presence. *Presence*, 1(1):120–126, 1992.

[SHH+03] N. Shibano, P. V. Hareesh, H. Hoshino, R. Kawamura, A. Yamamoto, M. Kashiwagi et K. Sawada K : Cyberdome : Pc clustered hemi spherical immersive projection display. *In Int Conf Artif Real Telexistence*, volume 13, pages 242–248, 2003.

[SK08] Toshiaki Suzuki et Kouichi Kawamoto : United states patent 7316617 : Game system and storage medium having stored thereon game program, January 2008.

[SL92] M. Soucy et D. Laurendeau : Multi-resolution surface modeling from multiple range views. *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on*, pages 348–353, 1992.

[Sla99] M. Slater : Measuring Presence : A Response to the Witmer and Singer Presence Questionnaire. *Presence*, 8(5):560–565, 1999.

[SLJ98] Nigel Stewart, Geoff Leach et Sabu John : An improved Z-buffer CSG rendering algorithm. *1998 Eurographics/Siggraph Workshop on Graphics Hardware*, pages 25–30, Aug 1998.

[SMDW98] Shamus Smith, Tim Marsh, David Duke et Peter Wright : Drowning in immersion. *In UK-VRSIG'98*, 1998.

[SS95] Linda Von Schweber et Erick Von Schweber : Virtual reality - virtually here, 1995.

[SS00] Mel Slater et Antony Steed : A virtual presence counter. *Presence*, 9(5):413–434, octobre 2000.

[STU07] M. Schwaiger, T. Thummel et H. Ulbrich : Cyberwalk : An advanced prototype of a belt array platform. *Haptic, Audio and Visual Environments and Games, 2007. HAVE 2007. IEEE International Workshop on*, pages 50–55, 2007.

[SU93] M. Slater et M. Usoh : Presence in immersive virtual environments. *Virtual Reality Annual International Symposium, 1993., 1993 IEEE*, pages 90–96, 1993.

[SUC95] M. Slater, M. Usoh et Y. Chrysanthou : The influence of dynamic shadows on presence in immersive virtual environments. *Virtual Environments*, 95:8–21, 1995.

[Sut65] Ivan E. Sutherland : The ultimate display. *In Proceedings of IFIPS Congress*, volume 2, pages 506–508, mai 1965.

[Sut68] Ivan E. SUTHERLAND : A head-mounted three-dimensionnal display. *In Proceedings of Fall Joint Computer Conference*, pages 757–764, 1968.

[SvdSKvdM01] Martijn J. SCHUEMIE, Peter van der STRAATEN, Merel KRIJN et Charles A.P.G. van der MAST : Research on presence in virtual reality : A survey. *CyberPsychology & Behavior*, 4(2):183–201, 2001.

[Tak06] Y. TAKAKI : High-density directional display for generating natural three-dimensional images. *Proceedings of the IEEE*, 94(3):654–663, 2006.

[TC89] C.H.O.H. TEH et R.T. CHIN : On the Detection of Dominant Points on Digital Curves. *IEEE Transactions on pattern analysis and machine intelligence*, 2(8):859, 1989.

[Tha93] D THALMANN : Using Virtual Reality Techniques in the Animation Process. *Virtual Reality Systems (Earnshaw R, Gigante M, Jones H eds), Academic Press*, pages 143–159, 1993.

[TL94] G. TURK et M. LEVOY : Zippered polygon meshes from range images. *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 311–318, 1994.

[TLMS03] C. THEOBALT, M. LI, M. MAGNOR et H. SEIDEL : A flexible and versatile studio for synchronized multi-view video recording, 2003.

[TR+03] B. THERE, G. RIVA *et al.* : Measuring Presence : Subjective, Behavioral and Physiological Methods. *Being There : Concepts, Effects and Measurements of User Presence in Synthetic Environments*, 2003.

[Tsa22] Roger Y. TSAI : A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. pages 221–244, 1922.

[TWHH02] Wai-Kwan TANG, Tien-Tsin WONG, Pheng-Ann HENG et Pheng-Ann HENG : The immersive cockpit. *In MULTIMEDIA'02 : Proceedings of the tenth ACM international conference on Multimedia*, pages 658–659, 2002.

[UAW+99] M. USOH, K. ARTHUR, M.C. WHITTON, R. BASTOS, A. STEED, M. SLATER et F.P. BROOKS JR : Walking> walking-in-place> flying, in virtual environments. *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 359–364, 1999.

[UCAS00] M. USOH, E. CATENA, S. ARMAN et M. SLATER : Using Presence Questionnaires in Reality. *Presence : Teleoperators & Virtual Environments*, 9(5):497–503, 2000.

[VO90] P. VUYLSTEKE et A. OOSTERLINCK : Range image acquisition with a single binary-encoded light pattern. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):148–164, 1990.

[WAB93] Colin WARE, Kevin ARTHUR et Kellogg S. BOOTH : Fish tank virtual reality. *In CHI '93 : Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*, pages 37–42. ACM, 1993.

[WBM+02] J.R. Wolpaw, N. Birbaumer, D.J. McFarland, G. Pfurtscheller et T.M. Vaughan : Brain–computer interfaces for communication and control. *Clinical Neurophysiology*, 113(6):767–791, 2002.

[Wid01] T. Widjanarko : Brief survey on three-dimensional displays : from our eyes to electronic hologram, 2001. http ://www-staff.lboro.ac.uk/ mmtw/holopaperWeb.pdf.

[Wie96] T. F. Wiegand : Interactive rendering of CSG models. *Computer Graphics Forum*, 15(4):249–261, 1996.

[Wil78] L. Williams : Casting curved shadows on curved surfaces. *ACM SIGGRAPH Computer Graphics*, 12(3):270–274, 1978.

[WK04] J. Woetzel et R. Koch : Multi-camera real-time depth estimation with discontinuity handling on pc graphics hardware. *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, 1:741–744, Aug. 2004.

[WLSG02] Stephan Wurmlin, Edouard Lamboray, Oliver G. Staadt et Markus H. Gross : 3d video recorder. *pg*, 00:325, 2002.

[Woo89] Robert J. Woodham : Photometric method for determining surface orientation from multiple images. pages 513–531, 1989.

[WS98] B.G. Witmer et M.J. Singer : Measuring Presence in Virtual Environments : A Presence Questionnaire. *Presence*, 7(3):225–240, 1998.

[WW03a] John A. Waterworth et Eva L. Waterworth : The meaning of presence. *In Presence connect*, 2003.

[WW03b] O. Wulf et B. Wagner : Fast 3D Scanning Methods for Laser Measurement Systems. *International Conference on Control Systems and Computer Science, Bucharest, Romania, July*, 2003.

[XQJ06] Li Xu, Feihu Qi et Renjie Jiang : Shadow removal from a single image. *Intelligent Systems Design and Applications, 2006. ISDA '06. Sixth International Conference on*, 2:1049–1054, Oct. 2006.

[YSK+02] Shuntaro Yamazaki, Ryusuke Sagawa, Hiroshi Kawasaki, Katsushi Ikeuchi et Masao Sakauchi : Microfacet billboarding. *In EGRW '02 : Proceedings of the 13th Eurographics workshop on Rendering*, pages 169–180, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[YWB02] Ruigang Yang, Greg Welch et Gary Bishop : Real-time consensus-based scene reconstruction using commodity graphics hardware. *In PG '02 : Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, page 225, Washington, DC, USA, 2002. IEEE Computer Society.

[ZA02] Z-A : La première salle immersive stéréo active transportable sur pc : Le sas^{CUBE}. Dossier de presse, 2002.

[Zha00a] Zhengyou ZHANG : A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

[Zha00b] Zhengyou ZHANG : A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

[ZJ05] Zhiwei ZHU et Qiang JI : Robust real-time eye detection and tracking under variable lighting conditions and various face orientations. *Comput. Vis. Image Underst.*, 98(1):124–154, 2005.

[ZKRB05] C. ZACH, K. KARNER, B. REITINGER et H. BISCHOF : Space Carving on 3D Graphics Hardware. Rapport technique #2004-013, VRVis, 2005.

A

Principales interfaces de réalité virtuelle

Pour que l'immersion dans un environnement virtuel puisse se faire, il faut que l'utilisateur agisse sur les différents éléments présents dans cet univers mais qu'il puisse aussi avoir un retour de ce dernier. Ces actions ne sont possibles qu'au travers d'un ensemble d'interfaces dont nous allons détailler les principales lignes.

A.1 La capture de position et de mouvement

Une bonne interaction entre l'environnement virtuel et un utilisateur passe par un suivi de la position de ce dernier par rapport au dispositif de restitution du système de réalité virtuelle. Différentes techniques de capture de position existantes vont être décrites dans cette section.

A.1.1 La capture optique

La première catégorie d'interfaces de suivi de position repose sur l'emploi de marqueurs visuels placés sur l'utilisateur que l'on associe à un système vidéo. Le principal avantage de l'utilisation de marqueurs réside dans leur taille, généralement petite, qui permet de les placer sur le corps de l'utilisateur sans pour autant que ce dernier se sente gêné. Les marqueurs peuvent se présenter sous la forme de petites boules qui réfléchissent la lumière émise par des sources infrarouges ou alors sous la forme de diodes infrarouges ou de couleur. Malgré cela, les mouvements de l'utilisateur peuvent entraîner des occlusions des marqueurs ce qui implique alors de multiplier le nombre de caméras pour couvrir le maximum d'espace. De plus, la spatialisation d'un marqueur nécessite que les caméras soit calibrées [Zha00b] ce qui représente une phase assez délicate et source d'imprécisions. Des systèmes ont été créés en conséquence pour limiter ce genre de limitation comme l'«*Optotrak*» de la société *Northern Digital*.

FIG. 1: Le matériel de l'*Optotrak Copyright : Northern Digital*

Certains matériels peuvent également voir leur usage traditionnel détourné pour créer une interface de *tracking* assez facilement. Ce cas peut s'illustrer avec la manette Wiimote [SK08] de Nintendo qui, une fois positionnée telle une caméra, permet d'effectuer le *tracking* de la tête d'un utilisateur [Lee07] qui porte sur des lunettes ou une casquette des sources de lumière infrarouge.

Il existe des systèmes de capture vidéo qui ne nécessitent pas que l'utilisateur ait à porter des marqueurs. Le principe est d'utiliser un grand nombre de caméras dans un environnement connu ce qui permet de faire une extraction des éléments mobiles de la scène puis de reconstruire le modèle correspondant afin d'en récupérer les caractéristiques [CGH05]. Une telle installation est également utilisée dans le domaine de la production télévisée ou cinématographique avec par exemple le dispositif «*CyberDome*» utilisé par la société *XD Production* qui permet d'effectuer une capture en temps réel.

A.1.2 La capture électromagnétique

La capture électromagnétique se fonde sur l'émission d'ondes magnétiques entre un émetteur et un ou plusieurs récepteurs à 6 degrés de liberté. En induisant successivement un courant électrique dans l'une des trois bobines de l'émetteur il va être possible depuis le récepteur qui va capter les ondes émises sur l'une de ses trois bobines. En fonction de l'intensité des ondes reçues et de l'orientation des bobines, il est possible de déterminer la position et l'orientation du récepteur par rapport à l'émetteur.

Avec ce système de capture, il devient plus simple d'effectuer un suivi de l'utilisateur car il n'y a pas de problème d'occlusion. Par contre l'utilisation d'un champ électromagnétique comme moyen de communication implique qu'aucun objet métallique ne soit présent dans l'environnement car il pourrait perturber les transmissions. Enfin les ondes n'ont pas un très grand champ d'action, ce qui confine leur utilisation à un espace relativement réduit ce qui peut être une forte contrainte dans les systèmes de réalité virtuelle qui tendent à s'agrandir.

Enfin il est intéressant de mentionner qu'il est aussi possible de calculer l'orientation d'un utilisateur en munissant ce dernier d'une boussole électronique qui va se référer au

FIG. 2: Le matériel du *Polhemus Copyright : Polhemus*

champ magnétique induit par la terre. Mais la contrainte d'objets métalliques présents autour de l'utilisateur pouvant influer la détection existe toujours.

A.1.3 La capture acoustique

Pour évaluer la position et l'orientation d'un utilisateur dans un système de réalité virtuelle, il existe une technique qui se fonde sur le temps que met à parcourir un son. Un émetteur fixe diffuse des ultrasons qui sont captés par un récepteur placé sur l'utilisateur. Pour calculer l'orientation de ce dernier, le système repose sur trois émetteurs et trois récepteurs placés aux sommets d'un triangle équilatéral, ce qui permet de faire une triangulation à partir des décalages temporels.

Par exemple, ce type de capture a été employé par *Logitech* dans le cadre de la création d'une souris sans fil. Toutefois, les contraintes sont assez importantes. Tout d'abord une stabilité de l'environnement est très importante car une simple variation du taux d'humidité peut entraîner un certain nombre d'erreurs de calculs. Ensuite, l'émetteur et le récepteur doivent se faire face dans la mesure du possible sans quoi la triangulation est impossible. Bien que le coût de ce système soit relativement faible, celui-ci n'a pas du tout connu de succès.

A.1.4 La capture mécanique

La précision dans le calcul d'un mouvement du corps n'est généralement obtenue que si l'utilisateur porte un appareillage mécanique permettant de connaître l'angle relatif entre chacune des articulations (codeurs angulaires ou potentiomètres). Ce type de système, appelé exosquelette, peut permettre d'obtenir soit la position de l'ensemble du corps soit uniquement certaines parties telles qu'un bras ou la tête. L'exosquelette est difficilement exploitable en réalité virtuelle car il oblige de porter une armature parfois lourde qui peut s'avérer difficilement manipulable. De plus ce genre de matériel n'est pas toujours adapté à toutes les morphologies.

Pour détecter un geste effectué avec les doigts, il existe une interface nommée «gant de données» qui, une fois enfilé, va déterminer l'angle dessiné par chacun des doigts ainsi que les positions relatives entre eux. On retrouve d'un côté des gants qui peuvent être constitués d'une armature mécanique mais qui dans ce cas impose une manipulation difficile, et de l'autre des gants basés sur la fibre optique qui sont plus fragiles mais moins intrusifs en raison de leur faible volume.

Enfin, une autre catégorie de capture mécanique est le *spidar* qui a pris ce nom à cause de l'aspect physique du système qui se compose de multiples câbles. Principalement utilisé dans le positionnement des doigts dans un espace défini, ce système fonctionne en rattachant chaque câble, lié à un moteur, à un doigt de l'utilisateur. Le recoupement des informations issues des différents moteurs permet alors d'obtenir la position d'un doigt.

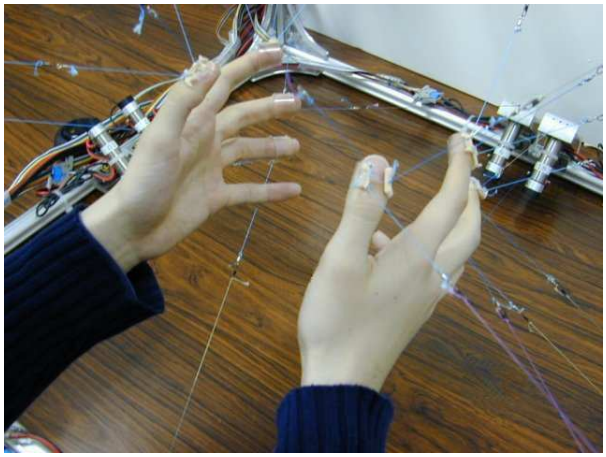


FIG. 3: Présentation du *spidar Copyright : Tokyo Institute of Technology*

A.1.5 Les *BCI*

Les *BCI* ou *Brain Computer Interfaces* [WBM+02] sont des capteurs qui permettent de récupérer des signaux émis par l'activité cérébrale. Après avoir subi un traitement, ces informations sont utilisées pour être interprétées sous la forme d'une commande ou en déduire une réaction de l'utilisateur. La difficulté réside principalement dans la traduction d'un signal en une donnée exploitable, car le cerveau reste pour le moment un organe relativement complexe. Les possibilités disponibles sont donc à l'heure actuelle assez restreintes mais permettent d'effectuer des interactions simples avec l'environnement virtuel [LLR+06].

Depuis peu, il existe des versions grand public des *BCI* (figure 4) qui sont proposées par la société *Emotiv* et destinées à être utilisées dans les jeux vidéo. Mais les possibilités sont très limitées pour le moment.

FIG. 4: Le *BCI* de Emotiv *Copyright : Emotiv*

A.2 Les interfaces haptiques

Une interface dite haptique implique toute interface qui a un lien avec le toucher, mais d'une manière plus étendue elle fait aussi intervenir le sens kinésthésique. Ce dernier regroupe toutes les sensations liées au mouvement des membres du corps qui sont décelables par la tension ou le relâchement des muscles ou alors liées à leur position (proprioception). Dans cette section nous allons voir les principaux mécanismes utilisés pour simuler ces sensations.

A.2.1 Le retour d'effort

Ce type d'interface est une des plus connue par le grand publique principalement à cause de sa popularisation par l'intermédiaire du jeu vidéo. Pour augmenter l'immersion d'une activité ludique, diverses interfaces, issues de multiples recherches en RV, ont été développées telles que les volants ou joysticks à retour d'effort. Ces outils ont l'avantage de rendre l'expérience de jeu plus crédible, voire plus réaliste.

La plupart des interfaces haptiques à retour d'effort fonctionnent selon le même principe : c'est un système physique qui vise à simuler un outil réel, utilisé par exemple dans le cadre de l'industrie, et qui va réagir en fonction de l'environnement virtuel. A cette fin, ces équipements sont munis de moteurs qui vont directement agir sur leurs articulations ou leurs axes de mouvement.

Ainsi, on retrouve ce type d'interface à retour d'effort sur des bras mécaniques qui se composent de quatre articulations car outre celles qui vont relier les trois membres, il y a celle qui permet d'appliquer une rotation du système sur 360°. Un système de câbles va contrôler les six degrés de liberté d'action accordés par l'interface permettant de manipuler un outil virtuel qui interagit avec d'autres éléments virtuels. On peut dire qu'il existe deux grandes catégories de bras : les grands, souvent destinés à un emploi en simulation industrielle (et souvent munis d'un gros bouton rouge en cas de dangereux emballement du bras) et les petits qui permettent un travaille plus fin tel que la sculpture, la manipu-

lation de géométries ou alors la simulation chirurgicale.

FIG. 5: Exemple de bras pneumatique à retour d'effort *Copyright : Southern Methodist University*

Parmi les autres interfaces à retour d'effort, certaines sont dédiées pour les doigts tels que les «gants» mécaniques qui permettent d'agir précisément sur chaque phalange, ou une version du «*spidar*» dont chaque câble est relié à un moteur.

A.2.2 Le retour tactile

Il existe certaines interfaces qui peuvent permettre à l'utilisateur d'avoir un retour tactile de l'environnement virtuel. Les doigts étant l'une des extrémités les plus sensibles de notre corps, la plupart des interfaces ont été conçues afin de stimuler le sens du toucher.

Il existe plusieurs techniques pour rendre l'impression de toucher avec plus ou moins d'efficacité et qui viennent s'ajouter à un gant de données. Une première consiste à placer une série de microballons qui peuvent être rapidement gonflés ou dégonflés au niveau des doigts et de la paume en fonction de ce qui est «touché» dans l'univers virtuel. Une autre interface tactile fait intervenir des matrices d'aiguilles qui varient selon la granularité de la surface et offrent ainsi l'avantage d'accentuer l'interaction à cause du grand nombre de points de contact possibles. Enfin, une dernière interface tactile consiste à provoquer des vibrations sur différentes parties de la main. On retrouve par extension ce genre de retour tactile sur certaines manettes de jeu qui vibrent lorsqu'un élément virtuel est touché.

Lorsque qu'on se déplace, il se provoque un retour tactile via le déplacement de l'air. Cela se concrétise, en réalité virtuelle, par l'utilisation de ventilateurs qui tournent plus ou moins vite selon l'impression de vitesse de déplacement que l'on cherche à reproduire.

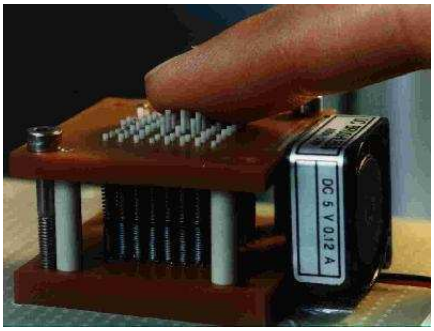


FIG. 6: Exemple de matrice d'aiguilles utilisée pour le retour tactile *Copyright : Forschungszentrum Karlsruhe*

A.2.3 Les simulateurs de mouvement

Il y a deux types de mouvements qui peuvent être simulés mais qui souffrent de la contrainte de l'espace limité de l'installation dans laquelle ils sont utilisés. Dans la première catégorie, nous trouvons les simulateurs de véhicules qui par l'utilisation d'une plate-forme disposée sur des vérins électriques qui permettent d'appliquer des rotations du corps (pour les virages par exemple) et des accélérations. Certains mouvements ne sont pas directement reproductibles via le simulateur comme celui du freinage car cela nécessiterait une cabine de simulation qui se déplace sur une grande surface, ce qui implique un coût de production assez important. Dans ce genre de cas des subterfuges sont utilisés afin de tromper les sens de l'utilisateur en associant par exemple un mouvement simple avec une action visuelle pour provoquer la réaction attendue.

L'autre catégorie de mouvements consiste à faire croire à l'utilisateur qu'il peut se déplacer en marchant dans l'environnement virtuel et qu'un pas dans le réel implique un pas dans le virtuel. La difficulté est double, il faut tout d'abord pouvoir détecter la volonté de l'utilisateur de vouloir se déplacer et ensuite, il faut faire en sorte que lorsque l'utilisateur se déplace, il puisse le faire de manière tout à fait naturelle. Parmi les interfaces de locomotion, on trouve tout d'abord les tapis 1D ou 2D qui permettent d'obtenir une surface en mouvement via un tapis roulant qui fait faire du surplace à l'utilisateur. Dans le cas d'un tapis 1D l'utilisateur n'a la possibilité de se déplacer que dans une direction donnée et ne pourra pas accomplir l'action physique associée au changement de direction. Cela ne devient réalisable que lors de l'utilisation d'un tapis 2D mais demande un développement plus complexe comme le «*CyberWalk*» [STU07] ou celui de la société «*Virtual Space Devices*». Certaines interfaces permettent même une rotation du tapis pour donner l'illusion que l'utilisateur est en train de grimper un pente. Toutefois, il est difficile d'associer la volonté de se déplacer de l'utilisateur au mouvement du tapis, ce qui fait que ce dernier impose généralement le mouvement ou est actionné via une commande. C'est cette limitation qui implique que l'utilisateur soit harnaché afin d'éviter tout accident lié à une perte d'équilibre.

Il existe un autre genre d'interface qui reprend un peu le principe de la roue de hamster

FIG. 7: Le *CyberWalk Copyright : Schwaiger et al.*

mais avec une sphère [LL06] : la sphère est posée sur un ensemble de patins afin qu'elle puisse tourner tout en restant sur place. De cette manière, chaque pas de l'utilisateur va entraîner une rotation de la sphère qui déclenchera un déplacement dans le monde virtuel. Outre le temps d'adaptation nécessaire à la maîtrise de cette interface (surtout lors des changements de vitesse de déplacement), la seule restitution visuelle qui semble utilisable est le HMD qui peut déstabiliser l'équilibre de l'utilisateur.

Une équipe de recherche de l'université Tsukuba au Japon a proposé deux types d'interface de locomotion. La première, nommée «Powered Shoes» [IYT06], est fondée sur un système de roulettes, commandées par de petits moteurs qui viennent se fixer sous les chaussures de l'utilisateur. Lorsqu'il se déplace ces roulettes se mettent en action dans la direction inverse du mouvement afin de le maintenir sur place. La seconde, «CirculaFloor» [IYFN05], utilise un système astucieux de dalles mobiles (figure 8) qui viennent se relayer sous les pieds de l'utilisateur tout en le maintenant sur place. Un jeu de quatre dalles suffit pour répondre à l'ensemble des possibilités de déplacement que fait l'utilisateur, mais uniquement dans un contexte de marche lente.

FIG. 8: Le *Circulafloor Copyright : Iwata et al.*

A.2.4 Le pseudo-haptique

Le pseudo-haptique ou illusion haptique est un mécanisme qui va tenter de se substituer à une interface haptique via la stimulation d'un autre sens. Cela passe généralement par la vue qui, comme cela a déjà été mentionné, est l'un des sens les plus utilisé chez l'être humain et qui en plus a la particularité de participer à la perception haptique de l'environnement (exemple typique des rochers en polystyrène utilisés pour le trucage au cinéma).

Lécuyer *et al.* [LCK+00] ont démontré que certaines actions effectuées par l'intermédiaire d'une simple souris voire d'une souris 3D, pouvaient provoquer un retour haptique. Il présente ainsi l'exemple d'une surface virtuelle [LBE04] sur laquelle sont placés de petits monticules ou creux qui vont influencer la vitesse de déplacement du curseur lorsque l'utilisateur le fait passer sur l'un d'eux. Pour évaluer l'effet, il cache dans un premier temps la surface puis la révèle et demande à chaque fois à l'utilisateur de déterminer la forme de la surface sur laquelle il vient de passer le curseur. Dans les deux cas, les résultats sont supérieurs à 85% d'exactitude dans la reconnaissance, ce qui démontre que sans retour haptique, il est quand même possible de le simuler.

A.3 Les autres interfaces

A.3.1 Les interfaces sonores

Deux familles d'interfaces de restitution sonore sont utilisées en réalité virtuelle. La sélection d'un procédé plutôt qu'un autre n'est pas forcement évidente car chacune d'elles à ses avantages et ses inconvénients.

Le premier type d'interfaces sont les écouteurs audio. Leur principal bénéfice est de pouvoir fournir une restitution sonore individuelle à coût réduit, tout en isolant l'utilisateur des différentes nuisance pouvant venir de l'extérieur. Comme la synthèse d'un son est binaurale, la difficulté est de pourvoir le spatialiser les différentes sources sonores virtuelles. Pour cela, il existe un ensemble de filtres qui ont été mis au point en utilisant une décomposition des *HRTF* (*Head Related Transfer Function*).

La seconde catégorie d'interfaces comprend l'ensemble des système de restitution reposant sur des enceintes. Ces dernières sont réparties de manière précise autour de l'espace dans lequel l'utilisateur est censé évolué. L'exemple le plus courant consiste à positionner les enceintes sur une sphère virtuelle au centre de laquelle est placé un individu. Cette configuration est la plus adaptée à la spatialisation car un son peut être émis de presque n'importe où dans l'espace. Mais, contrairement aux écouteurs, les enceintes sont destinées à un seul utilisateur qui est censé rester en un point précis et qui peut être dérangé par des bruits extérieurs. De plus, ce type d'installation requiert une mise en place minutieuse qui peut s'avérer coûteuse.

Quel que soit la famille à laquelle appartient l'interface utilisée, il faut quand même compter sur un *tracking* de l'utilisateur afin de pouvoir corriger en temps réel le produit des différentes sources sonores. La spatialisation n'est à priori possible que de cette manière si on considère que l'utilisateur est libre de se déplace dans l'installation.

A.3.2 Les interfaces olfactives

Les interfaces olfactives sont probablement celles qui sont le moins répandues. Les raisons sont doubles : il tout d'abord difficile de délivrer tout le panel d'odeurs qu'il est possible de rencontrer au quotidien, ensuite lorsqu'une odeur est émise il n'est pas évident de la faire disparaître de manière instantanée comme cela est nécessaire lors d'un déplacement rapide.

Parmi les interfaces existantes, nous pouvons mentionner celle proposée par la société *Olfacom* qui permet de diffuser des arômes depuis des cartouches (figure 9). Un autre type d'interface a été développé par *Aromajet* qui propose des applications au jeu vidéo et à la médecine.

FIG. 9: Une interface de restitution olfactive *Copyright : Olfacom*

B

Création d'une image stéréoscopique avec *OpenGL*

Quelque soit le système utilisé (photographie, informatique,...), le processus de création d'une image stéréoscopique reste globalement le même : gnérer deux images d'une même scène à partir de deux points de vue légèrement décalés.

Nous allons voir qu'avec *OpenGL*, il est possible de créer des images stéréoscopiques et qu'il n'existe pas une méthode unique.

B.1 Le plan de convergence

Le plan de convergence est un plan fictif dont le positionnement dans l'environnement virtuel va influencer notre perception. On définit ce plan comme étant la limite qui va donner une impression de relief ou une impression de profondeur. Au contraire, tout élément de la scène se situant au niveau de ce plan apparaîtra au niveau de la surface du support d'affichage. Selon sa zone, initiale un point peut se projeter de trois manières différentes. On parle alors de parallaxe positive (figure 1a), parallaxe négative (figure 1b) ou parallaxe nulle (figure 1c).

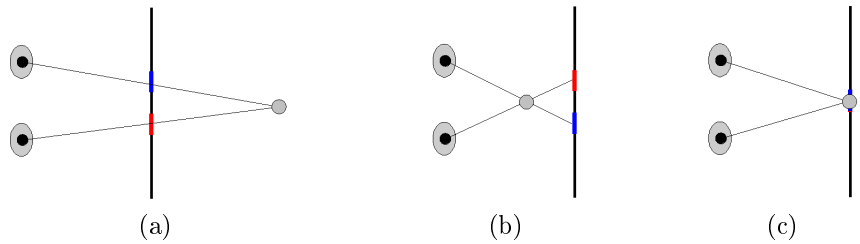


FIG. 1: Les trois catégories de parallaxe : *(a)* parallaxe positive *(b)* parallaxe négative *(c)* parallaxe nulle

B.2 La méthode *toe-in*

Le principe de la méthode d'images stéréoscopiques nommée *toe-in*, qui pourrait être traduite comme «focalisation des points de visée», consiste à faire en sorte que la vue de droite et la vue de gauche regardent un même point de l'espace. Les positions de l'œil droit et de l'œil gauche dans la scène *OpenGL* sont calculées en respectant une distance qui les sépare de *dio* valant en moyenne $6,5cm$.

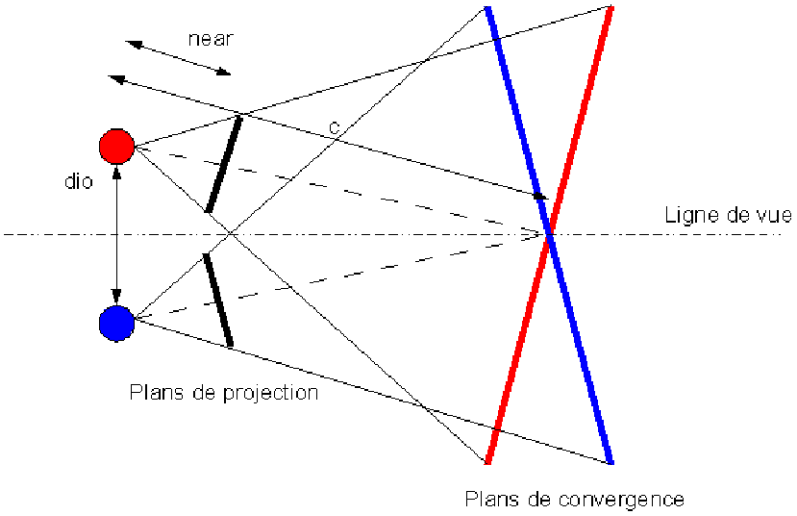


FIG. 2: Illustration de la méthode *toe in*

La position du point de visée dans la scène va définir le plan de convergence (voir la figure 2). Ce point appartient obligatoirement à la droite perpendiculaire au segment *dio* et passant par son centre. Pour créer une image stéréoscopique fondée sur le procédé de restitution rouge-cyan, il est nécessaire de suivre les étapes suivantes :

Listing B.3: Code de la méthode *toe-in*

```
1
2  float right[3]; // position du point de vue droit
3  float left[3]; // position du point de vue gauche
4  float focus[3]; // position du plan image
5  ...
6  glMatrixMode(GL_PROJECTION);
7  glLoadIdentity();
8  gluPerspective(60,(double)screenwidth/(double)screenheight,0.1,100.0);
9
10 glMatrixMode(GL_MODELVIEW);
11 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
12 glLoadIdentity();
13 // Rendu de la couleur rouge
```

```
14 glColorMask(GL_TRUE,GL_FALSE,GL_FALSE,GL_TRUE);
15 gluLookAt(left[0],left[1],left[2],
16              focus[0],focus[1],focus[2],
17              0.0,1.0,0.0);
18 draw();
19
20 glClear(GL_DEPTH_BUFFER_BIT);
21 glLoadIdentity();
22 // Rendu de la couleur cyan
23 glColorMask(GL_FALSE,GL_TRUE,GL_TRUE,GL_TRUE);
24 gluLookAt(right[0],right[1],right[2],
25              focus[0],focus[1],focus[2],
26              0.0,1.0,0.0);
27 draw();
28 glColorMask(GL_TRUE,GL_TRUE,GL_TRUE,GL_TRUE);
29 ...
```

Malgré sa grande simplicité de mise en œuvre, cette méthode souffre de problèmes de parallaxe verticale et horizontale sur les bords de l'image. Ceux-ci sont dus au fait que les plans de convergence associés aux deux points de vue ne sont pas parallèles, ce qui va gêner la perception du relief. Cette limite est illustrée sur la figure 3.

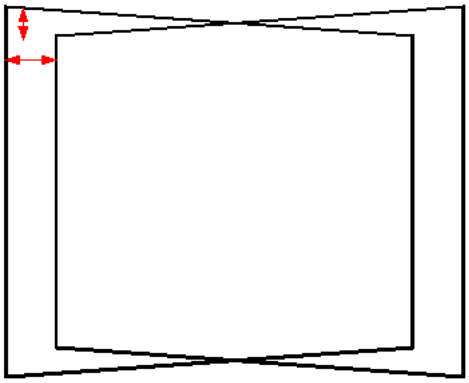


FIG. 3: Illustration des problèmes de parallaxes induits par la méthode *toe in*

B.3 La méthode *off axis*

La méthode de stéréoscopie, dite *off axis*, permet de régler le problème de parallaxe verticale en faisant en sorte que les plans de convergence associés aux deux points de vue soient parallèles. La méthode va même encore plus loin en ne définissant plus qu'un seul plan de convergence commun aux deux vues. La différence des points de vue va donc entraîner une modification des volumes de projection qui ne sont alors plus parallèles mais

asymétriques. La principale difficulté de la méthode consiste donc à définir les paramètres des plans de projection pour l'œil droit et ceux pour l'œil gauche.

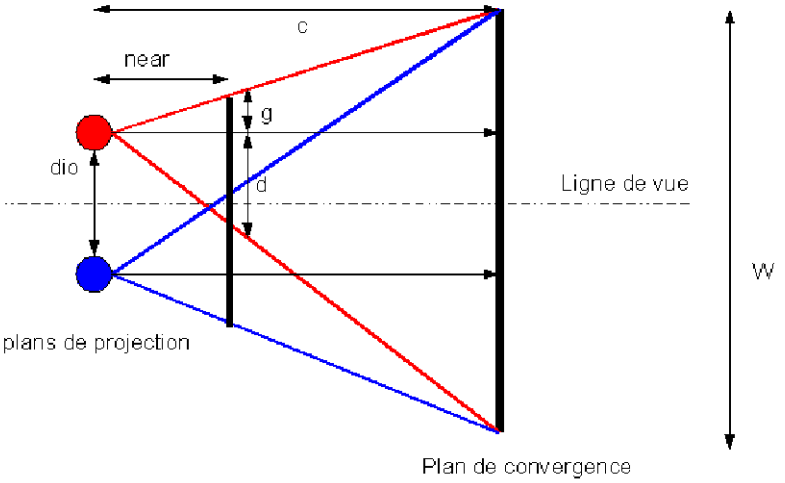


FIG. 4: Géométrie de la stéréoscopie *off axis*

Avec *OpenGL*, un plan de projection est défini par six paramètres dont les quatre premiers définissent les coordonnées des points inférieur gauche ($left$, $bottom$) et supérieur droit ($right$, top) par rapport au point principal. En utilisant les relations géométriques de la figure 4, ont peut en déduire la valeur de ces paramètres pour l'œil gauche :

$$\begin{array}{l} H = 2.0 \times c \times \tan(\theta/2.0) \\ W = H \times ratio \\ right = \frac{near}{c} \times \frac{W+dio}{2.0} \\ left = -\frac{near}{c} \times \frac{W-dio}{2.0} \\ \frac{top=H}{2.0} \\ bottom = -top \end{array} \tag{B.1}$$

et pour l'œil droit :

$$\begin{array}{l} H = 2.0 \times c \times \tan(\theta/2.0) \\ W = H \times ratio \\ right = \frac{near}{c} \times \frac{W-dio}{2.0} \\ left = -\frac{near}{c} \times \frac{W+dio}{2.0} \\ \frac{top=H}{2.0} \\ bottom = -top \end{array} \tag{B.2}$$

Le code *OpenGL*, pour créer une image stéréoscopique rouge-cyan avec la méthode *off axis*, est donné dans le listing B.4.

Listing B.4: Code de la méthode *toe-in*

```
1
2 float rightview[3]; // position du point de vue droit
```

```
3  float leftview[3]; // position du point de vue gauche
4  float direction[3]; // vecteur de direction de la vue
5  ...
6  glMatrixMode(GL_PROJECTION);
7  glLoadIdentity();
8  glFrustum(left_left,right_left,bottom,top,near,far);
9  glMatrixMode(GL_MODELVIEW);
10
11 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
12 glLoadIdentity();
13 // Rendu de la couleur rouge
14 glColorMask(GL_TRUE,GL_FALSE,GL_FALSE,GL_TRUE);
15 gluLookAt(leftview[0],leftview[1],leftview[2],
16           direction[0],direction[1],direction[2],
17           0.0,1.0,0.0);
18 draw();
19
20 glMatrixMode(GL_PROJECTION);
21 glLoadIdentity();
22 glFrustum(left_right,right_right,bottom,top,near,far);
23 glMatrixMode(GL_MODELVIEW);
24
25 glClear(GL_DEPTH_BUFFER_BIT);
26 glLoadIdentity();
27 // Rendu de la couleur cyan
28 glColorMask(GL_FALSE,GL_TRUE,GL_TRUE,GL_TRUE);
29 gluLookAt(rightview[0],rightview[1],rightview[2],
30           direction[0],direction[1],direction[2],
31           0.0,1.0,0.0);
32 draw();
33 glColorMask(GL_TRUE,GL_TRUE,GL_TRUE,GL_TRUE);
34 ...
```

C

Les *shaders*

C.1 Le processus de restitution d'*OpenGL*

La plupart des implémentations d'*OpenGL* intègrent une série d'étapes de traitement regroupées sous le terme de «*OpenGL rendering pipeline*» dont un résumé est présenté dans le schéma suivant :

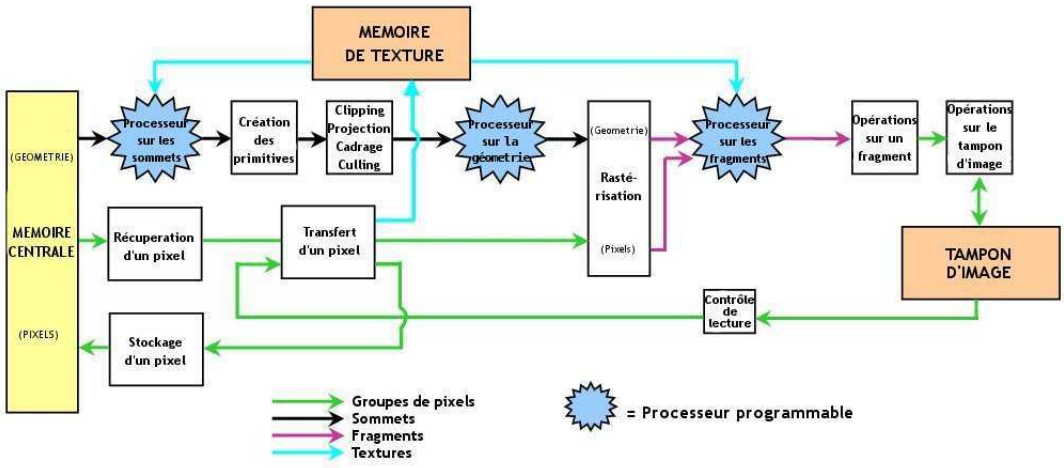


FIG. 1: Présentation du pipeline *OpenGL*

La première étape consiste à convertir un ensemble de sommets (points dans l'espace tridimensionnel) en primitives (points, lignes, polygones...). Dans un même temps, les coordonnées de texture sont transformées via la matrice de texture et le modèle d'illumination est appliqué aux sommets.

En parallèle, des données présentes dans la mémoire de l'ordinateur, sous forme de tableaux, sont lues pour être intégrées au processus de restitution. Ces données peuvent

être utilisées, par exemple, pour la création de textures ou alors pour mettre en correspondance un ensemble de valeurs chromatiques avec un rectangle de pixels de l'écran.

L'étape d'assemblage des primitives va permettre d'éliminer les primitives étant hors de l'espace d'affichage. Lors d'un cas limite, c'est à dire un cas dans lequel une primitive est à la frontière de la zone d'affichage, cette dernière est découpée en conséquence.

La rastérization est la conversion simultanée de données géométriques et de pixels en fragments. Chaque fragment correspond à un seul pixel et possède des informations telles qu'une valeur de profondeur, des valeurs chromatiques... Dans l'étape qui suit la rastérization, les fragments subissent une série d'opérations qui permettent de les comparer entre eux (test de profondeur, stencil...) et de leur associer les textures correspondantes.

La dernière étape correspond au rendu final, les fragments ayant passé l'étape précédente sont stockés dans un tampon qui sera visible à l'écran. On ne parle alors plus de fragment mais de pixel.

C.2 Le processeur graphique (GPU)

De plus en plus utilisés, les processeurs des cartes graphiques sont des outils extrêmement performants (de l'ordre de 500 à 700 transistors) qui peuvent travailler en parallèle du processeur principal. Ils peuvent donc apporter, par le biais des shaders qui sont des programmes spécifiques aux processeurs graphiques, un gain de performance non négligeable. Outre cet avantage, les cartes graphiques ont une architecture paralléle ce qui fait que les traitements sont beaucoup plus rapides. Toutefois le nombre de *pipelines* ne définit pas l'efficacité de la carte car il faut aussi lui associer la fréquence des processeurs. Par exemple la carte *nVidia 9600GT* possède 64 *pipelines* cadencé à une fréquence de 1625Hz tandis que la carte *ATI Radeon HD3870* a 320 *pipelines* cadencés à 775Hz.

Ces dernières années quelques optimisations ont fait leur apparitions. Tout d'abord une architecture unifiée a fait son apparition sur les cartes. Cela signifie que les différents programmes sur les shaders travaillent chacun de leur côté mais de manière uniforme signifiant que la charge de calculs peut être répartie plus facilement, améliorerant ainsi l'efficacité des GPUs. L'autre nouveauté consiste en l'utilisation de plusieurs cartes graphiques ce qui permet de multiplier les unités de calcul. On retrouve cette technologie sous le nom de *SLI* chez *nVidia* et de *CrossFire* chez *ATI*. Enfin la dernière nouveauté réside dans l'apparition de cartes graphiques bi-processeurs mais ces dernières souffrent actuellement de problèmes techniques qui limitent très fortement leurs capacités au vu de leurs possibilités théoriques.

C.2.1 Principe

Comme nous l'avons vu précédemment, le pipeline d'*OpenGL* effectue des opérations sur les sommets et sur les fragments. Les shaders permettent de redéfinir temporairement, par le biais de programmes, une partie de ces opérations effectuées en privilégiant l'utilisation du processeur de la carte graphique plutôt que le processeur central d'où le gain de performances. Il y a deux types de programmes, ayant une syntaxe proche du C, qui composent les shaders.

Le programme sur les sommets (ou *vertex shader*) s'emploie au niveau des opérations sur les sommets. Il reçoit un sommet sur lequel il va effectuer des transformations telles qu'appliquer la matrice de *modelview* ou de projection. Il est également possible de modifier sa couleur ou les coordonnées de texture qui lui ont été associées.

Le programme sur les pixels (ou *fragment shader*) s'utilise au niveau des opérations sur les fragments. Les étapes que ce programme remplace sont l'application de texture, la sommation des couleurs et le calcul du brouillard. D'autres opérations comme le test de profondeur, le test de scission, le test de stencil ou le test alpha ne sont pas modifiées. Son rôle est de définir les caractéristiques finales des fragments.

Depuis fin 2006 un nouveau programme a fait son apparition sous la dénomination de *geometry shader*. Comme son nom l'indique, ce programme permet d'appliquer un certain nombre d'opérations sur les sommets d'une géométrie (triangle, ligne ou point) avant qu'elle ne subisse l'étape de la rasterisation. Il devient alors possible de dupliquer les sommets, les transformer ou les supprimer. Comme c'est un *shader* récent, toutes les cartes graphiques ne sont pas capables d'être en mesure de les utiliser. Plus particulièrement, seules les cartes de chez *nVidia* à partir de la famille des 8XXX en ont la possibilité. Il est possible de vérifier cela en constatant la présence de l'extension *GL_EXT_geometry_shader4*.

C.2.2 Les différents langages

En fonction d'une API (*Application Programming Interface*) ou d'un constructeur de cartes graphiques, des langages spécifiques ont été conçus.

Cg [FK03] est un langage de haut niveau qui a été développé par le constructeur de carte graphique nVidia et est spécialement dédié aux produits de la même marque (son fonctionnement est limité sur les autres cartes graphiques). Ce langage repose sur l'utilisation d'un compilateur qui permet d'obtenir un code assembleur à partir d'un code écrit en *Cg*. Le code traduit est ensuite assemblé par les drivers de la carte graphique pour être utilisé dans l'application en cours de fonctionnement. Pour pouvoir utiliser le langage *Cg*, il faut posséder les extensions OpenGL suivante *GL_ARB_fragment_program* et *GL_ARB_vertex_program*.

GLSL [Ros04] ou *OpenGL Shading Language* est quant à lui dédié à OpenGL (à partir de la version 1.4). Le code source est envoyé à l'application OpenGL qui le compile, crée un exécutable (qui consiste à lier les shaders entre eux) et l'installe dans le processus de rendu. Toutes ces étapes se font à l'aide de fonctions *ARB* (qui seront probablement intégrées dans *OpenGL* 2.0 []). L'avantage de ce langage est qu'il peut être utilisé sur n'importe quelle carte graphique qui prend en charge les extensions suivantes : *GL_ARB_fragment_shader* et *GL_ARB_vertex_shader*. A l'heure actuelle, les cartes graphiques du type *ATI Radeon* 9XXX ne peuvent utiliser GLSL que sous un environnement Windows car la version d'OpenGL sous Unix pour les drivers *ATI* est encore 1.3.

HLSL [Eng04] est spécifique à l'API *Direct3D* (sous-couche, permettant la création de scènes 3D, de l'API *DirectX* de *Microsoft*) fonctionne sur le même principe de *GLSL* à la différence près que le compilateur n'est pas intégré dans les drivers.

TITRE : Approches visuelles pour l'amélioration de la présence en réalité virtuelle

NOM : de Sorbier de Pougnadoresse François

DISCIPLINE : Informatique

MOTS-CLÉS : réalité virtuelle, présence, immersion, stéréoscopie, flou, avatar

RÉSUMÉ :

Le sentiment de présence, but ultime de la réalité virtuelle, peut être atteint en stimulant ces quatre «piliers» que sont l'immersion, l'interaction, le maintien de la boucle action-perception et les émotions. Notre objectif est de proposer des méthodes visant à améliorer ce sentiment en s'intéressant plus particulièrement à la perception visuelle.

Dans cette optique, nous proposons tout d'abord une solution appliquant le rendu stéréoscopique sur carte graphique. Traditionnellement effectué en deux passes, ce rendu se fait maintenant en une passe, grâce aux *shaders* et au regroupement de certaines phases de calculs. Nous étendons ce processus de rendu aux tout récents écrans auto-stéréoscopiques nécessitant plus de deux vues, améliorant d'autant plus les temps de calcul. Pour assurer l'immersion et l'interaction, voire l'émotion, nous avons aussi cherché à diminuer la fatigue oculaire induite par les images stéréoscopiques, en ajoutant un flou de profondeur de champ. Ce flou, obtenu en temps réel grâce aux *shaders*, permet également d'inviter l'observateur à focaliser son attention sur des objets précis au lieu de laisser son regard errer.

Enfin, un objectif pour obtenir le sentiment de présence est de faire croire à l'utilisateur qu'il existe dans la scène virtuelle. Notre contribution à ce but, est d'intégrer de manière naturelle une représentation virtuelle de l'utilisateur. Pour cela, nous créons par *visual hulls* un avatar à l'aide de caméras. Finalement, cet avatar est employé pour illustrer la présence de l'utilisateur au travers de surfaces réfléchissantes virtuelles ou de la projection de son ombre.

Laboratoire d'Informatique de l'Institut Gaspard-Monge
UMR CNRS 8049
Université Paris-Est
Cité Descartes
5, Boulevard Descartes
Champs-sur-Marne
77454 Marne-la-Vallée Cedex 2